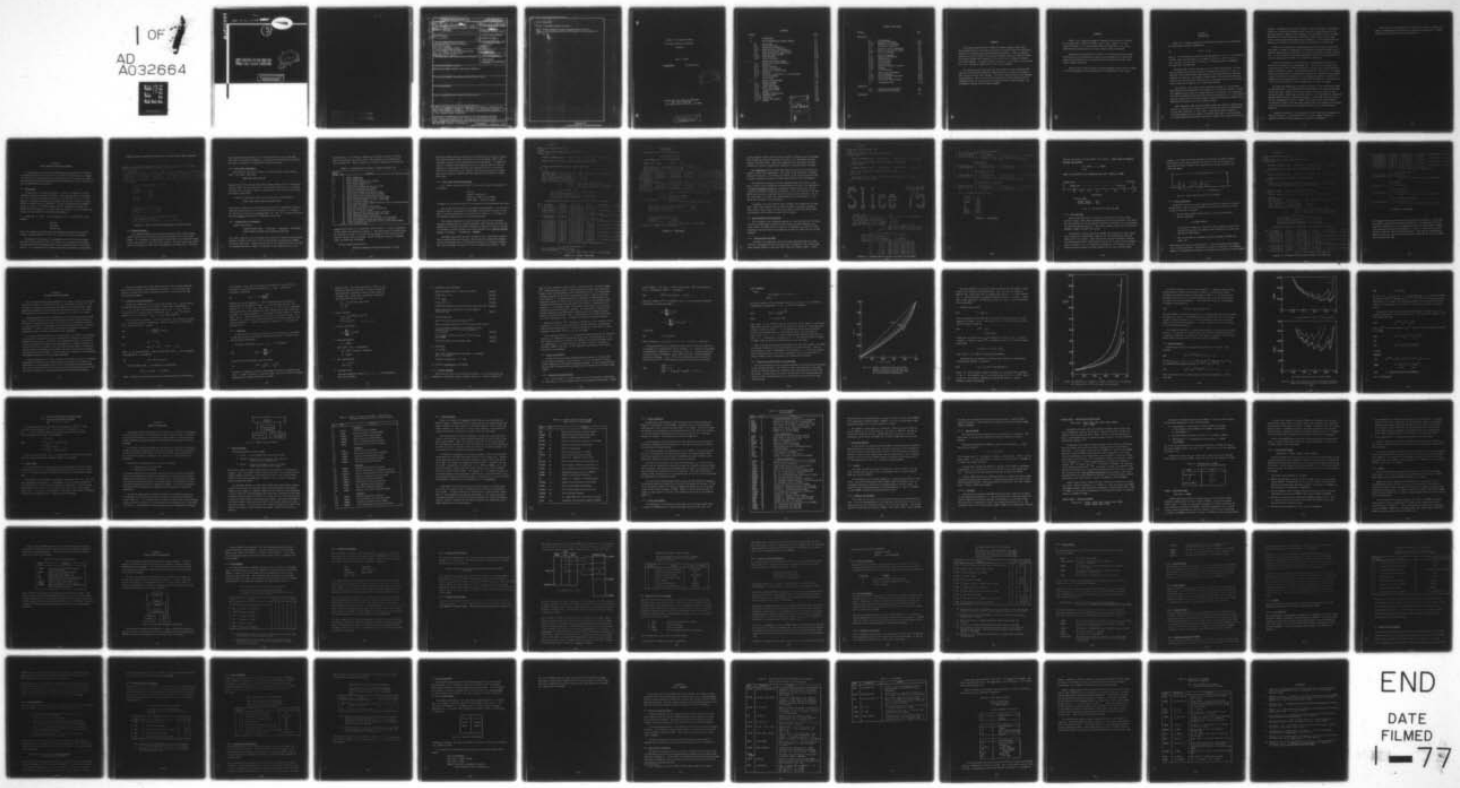
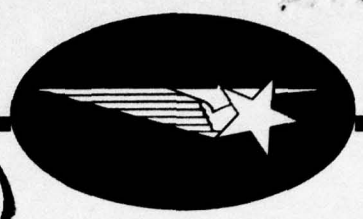
AD-A032 664 LOCKHEED MISSILES AND SPACE CO INC PALO ALTO CALIF PA--ETC F/G 9/2
SLICE 75, A COMPUTER PROGRAM FOR LARGE, SYMMETRIC EIGENVALUE PR--ETC(U)
SEP 76 P S JENSEN F44620-71-C-0109
UNCLASSIFIED LMSC/D508885 AFOSR-TR-76-1144 NL

1 OF 2
AD A032664



END
DATE FILMED
1-77

AFOSR - TR - 76 - 1144



(2)

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is
approved for public release IAW AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR - TR - 76 - 1144	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: LMSC/D508885	
4. TITLE (and Subtitle): SLICE 75, A COMPUTER PROGRAM FOR LARGE, SYMMETRIC EIGENVALUE PROBLEMS	5. TYPE OF REPORT & PERIOD COVERED: Final
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Paul S. Jensen	8. CONTRACT OR GRANT NUMBER(s): F44620-71-C-0109
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Lockheed Palo Alto Research Laboratory, 3251 Hanover Street, Palo Alto, California 94304	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F, 2304/A3
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research (NM), Bldg 410, Bolling Air Force Base, Washington, D. C. 20332	12. REPORT DATE: September 1976
	13. NUMBER OF PAGES: 79 (150 pp.)
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SLICE 75 is a computer program for eigenvalue and eigenvector analysis of the generalized, symmetric problem S x + Mx, where M is non-negative definite. It is particularly suited for obtaining a few select eigenvalues and eigenvectors of a large, sparse system of equations.

Comprehensive documentation of SLICE 75 is provided including detailed descriptions of the mathematical, numerical and implementation systems used. Operational procedures are also discussed and illustrated with cont.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 210 118

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20/Abstract

output from sample program executions.

SLICe 75 uses a general storage management system AID and a number of utility programs which are documented in the Appendixes of this report.

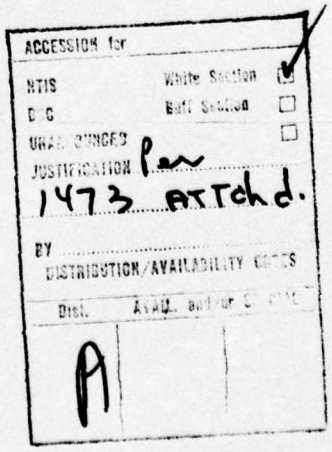
SLICE 75, A Computer Program For Large, Symmetric Eigenvalue Problems

Paul S. Jensen

LMSC-D508885 September 1976

Lockheed Palo Alto Research Laboratory
Dept. 5233, Bldg. 205
3251 Hanover Street, Palo Alto, CA 94304

CONTENTS



CONTENTS (Continued)

FOREWORD

This manual presents one of a number of results obtained from a five-year research effort on the numerical solution of elliptic partial differential equations by arbitrary grid finite difference techniques. Under the sponsorship of the Air Force Office of Scientific Research, Contract F44620-71-C-0109, this overall project was initiated in 1971 with Lt. Col. Nicholas P. Callas as contract manager. In August, 1973, the management of this contract was assumed by Lt. Col. Enrique H. Ramirez, who is the current manager.

The theoretical work that went into the extension of the sectioning technique for the generalized symmetric algorithm and the writing of this manual were carried out under this contract. The actual implementation of the computer program SLICE 75, which is based on the extended sectioning algorithm and is documented here, was sponsored by the Lockheed Missiles & Space Company, Inc. independent research and development program.

ABSTRACT

SLICE 75 is a computer program for eigenvalue and eigenvector analysis of the generalized, symmetric problem $S \underset{\sim}{x} = \lambda M \underset{\sim}{x}$, where M is non-negative definite. It is particularly suited for obtaining a few, select eigenvalues and eigenvectors of a large, sparse system of equations.

Comprehensive documentation of SLICE 75 is provided including detailed descriptions of the mathematical, numerical and implementation systems used. Operational procedures are also discussed and illustrated with output from sample program executions.

SLICE 75 uses a general purpose storage management system AID and a number of utility programs which are documented in the Appendixes of this report.

Section 1
INTRODUCTION

SLICE 75 is a computer program for determining eigenvalues and eigenvectors for the general, symmetric eigenproblem.

$$S \underset{\sim}{x} = \lambda M \underset{\sim}{x} \quad , \tag{1}$$

where M is non-negative definite and the null spaces of S and M are disjoint. In other words, for any vector $\underset{\sim}{y}$ we require $\underset{\sim}{y}^T M \underset{\sim}{y} \geq 0$ and if $M \underset{\sim}{y} = 0$, then $S \underset{\sim}{y} \neq \underset{\sim}{0}$ and vice versa. Of course, M positive definite immediately satisfies these requirements.

Although SLICE 75 can be used for (1) with various matrix forms, it is designed primarily for efficient analysis for large sparse matrices S and M. Typically, such problems result from discrete, variational analysis of partial differential equations. For example, finite element and finite difference programs for structural analysis make heavy use of eigenvalue analyses to determine natural modes and frequencies, and in buckling analysis.

A key feature of the large, sparse eigenvalue problem (1) is that only a small number of the eigenvalues and eigenvectors are generally required. Because the computation cost tends to be rather high for such problems, it is essential that this key feature be exploited by the solution algorithm. It is this requirement that renders most of the vast number of eigenvalue programs available (e.g., EISPACK [2]) unsuitable for large, sparse problems.

When eigenvalues from arbitrary regions within the eigenvalue spectrum are required, it appears that variants of the simultaneous iteration algorithm [8] are the most economical, see also [10]. A particular variant called the sectioning method [4] has a number of features which make it very attractive for large problems, particularly when several regions of the spectrum are to be

analyzed. Probably the key inherent feature is that it provides an exhaustive analysis of a specified contiguous region (section), taking advantage of the specified bounds to optimize the efficiency. In application of the sectioning algorithm, one typically analyzes a region and, based on the results, then selects other regions (often adjacent) for analysis. In this semi-interactive fashion, one can gain considerable insight to a problem at a minimal cost.

SLICE 75 is a comprehensive implementation of the sectioning method which incorporates a number of extensions to the original concepts presented in [4]. Important among these are simultaneous iteration and Chebyshev iterate acceleration. A brief overview of the sectioning method along with the mathematical extensions incorporated in SLICE 75 is presented in Section 3.

Besides the mathematical considerations, a program such as SLICE 75 is faced with a substantial data management task. The sizes of the matrices involved necessitate their being partitioned into blocks, most of which normally reside in auxiliary storage. The processing involves "simultaneous" processing of sets of large vectors, including the selection and accumulation of converged vectors which further complicates the data management problems. Probably the toughest blow to program simplicity is the fact that SLICE 75 is intended to be a form of utility program to solve problems from a variety of sources.

To design one program that could be very general in this environment seems an impossible task. So instead, SLICE 75 is designed as an executive processor which calls upon a system of utility processors to carry out specific data processing tasks. Supporting the entire system is an independent data manager [6] designed specifically for numerical processes. In this way, SLICE 75 is practically independent of the size, format and structure of the data involved. Utilization of SLICE 75 with new data structures thus only requires the construction of the appropriate utility programs with little or no modification of SLICE 75.

A general discussion of the structure of SLICE 75 and how it operates appears in Section 4. A brief description of the data manager appears in Appendix A and the utility programs are described in Appendix B.

The mechanics of using SLICE 75 are discussed in Section 2 along with a number of "good practice" heuristics for obtaining top efficiency. Several results from test cases are also provided there.

Section 2
USER INFORMATION AND SAMPLE RESULTS

In this section we provide basic information on the operation of SLICE 75 in sufficient detail for a user to solve a problem with the program. A number of options that a sophisticated user can exercise are not presented here since they require a substantial familiarity with the mathematics involved (Section 3) and the implementation (Section 4).

2.1 Matrix Data

To begin with, the symmetric matrices S and M of Equation (1) must somehow be generated external to SLICE 75. These matrices must be catalogued in a file via the data manager AID, which is used by SLICE 75 to access those matrices. An example processor which produces an AID file of matrices called MTXGEN is provided with the SLICE code. A user can generally modify MTXGEN to access S and M, in whatever fashion they are available, and simply let MTXGEN do the cataloguing. If a user wishes to construct a new cataloguing processor, the requirements are discussed in Appendix A.

At the time S and M are catalogued, the user is provided with three numbers:

File ID,
Group ID,
Contents ID.

Those three numbers provide SLICE 75 with sufficient information to extract records, literally named "S" and "M", via the file table of contents.

In Example 2.1 the results of cataloguing test 25 by 25 matrices along with other records on device AUX 1 are shown. The eigenvalues listed define the spectrum of the test problem which is used throughout Section 2 for illustration. It is constructed by performing similarity transforms on appropriate

diagonal matrices using factors obtained via a pseudo-random number generator.

```
 ENTER SIZE, PRIORITY AND PRINT OPTIONS FOR S, M, EIG VECS, EIG VALS,
 AND IF THE EIG VECS. ARE TO BE SAVED (2I3,1X,5L1)
?  25 10 FFFTF
     VECTOR EIG. VALS.  (   12293)  TYPE ( 0,1,0), DIMENSION  25, PRIORITY 10
  4.400000E-01  5.000000E-01  5.652217E-01  6.368364E-01  7.142856E-01  8.000000E-01
  8.947377E-01  1.000000E+00  1.117655E+00  1.250000E+00  1.400000E+00  1.571428E+00
  1.769923E+00  2.000000E+00  2.272728E+00  2.600000E+00  3.000000E+00  3.500000E+00
  4.142888E+00  5.000000E+00  6.200000E+00  8.000000E+00  1.100000E+01  1.700000E+01
  3.500000E+01

          THE  5 PRIMARY RECORD ID'S ARE

          RECORD          ID
          ----------------------
          CNTNTS         4097
          S              8146
          M              8195
          E VECS            0
          E VALS        12293


          ------------------------------
           FILE ESTABLISHED ON    AUX1
           FILE ID               90753
          ------------------------------

          ------------------------------------------------
            THE MTXGEN DATA OUTPUT GROUP ID     10241
            THE CONTENTS RECORD ID               4097
          ------------------------------------------------
```

Example 2.1 Sample Output from Cataloguing of Matrices S and M

2.2 Program Execution

For a beginner, SLICE 75 is most easily executed from a terminal since SLICE 75 asks for the information required and accepts it in a free field format. It is, however, a simple matter to have SLICE 75 read the small amount of required input data from a data file or cards. The basic input data is discussed in Sections 2.2.1 and 2.2.2 and summarized in Section 2.2.3. For more

complicated execution options, it is a little difficult to set up the data file without the SLICE 75 prompting. A capability for reading input driven by key words is expected to be implemented in the near future.

2.2.1 Data Access Information

After SLICE 75 goes into execution, it opens auxiliary storage devices "AUX 1" and "AUX 2" and prints

```
ENTER THE INPUT FIELD ID
?
```

After the File ID is entered, followed by carriage return, SLICE 75 immediately searches "AUX 1" for the named file. If it does not find the file, it repeats the above File ID request. After the third failure to find the requested file, the program simply stops.

If SLICE 75 successfully finds the requested file, it then prints

```
ENTER INPUT MATRIX GROUP AND CONTENT ID'S
?
```

After these ID numbers are entered (free field), SLICE 75 immediately finds the specified record group and the corresponding table of contents. It searches the contents for records with external names "S" and "M" and notes the corresponding internal record names for subsequent processing.

2.2.2 Program Control Information

SLICE 75 then prints

```
PROGRAM CONTROL FLAGS   11110 01001   11000 00000   00001 00000
ENTER INDEXES OF FLAGS TO BE SWITCHED ON ONE LINE
?
```

The control flags are a set of switches that control the detailed processing and printout of SLICE 75 and will be left in the default setting shown by simply typing carriage return. Should the user wish to flip switches such as 11, 6 and 2 (counting from left to right), for example, he may simply enter 11, 2, 6,

carriage return. If a switch is flipped an even number of times, its final state is the same as its initial state. The functions of the switches are listed in Table 2.1. Most of them are for detailed "debug" type printout and

Table 2.1 Program Control Switches (1 indicates default on)

Default State	No.	Function
1	1	Print eigenvalues
1	2	Print eigenvectors
1	3	Print elapsed computer time history
1	4	Print initial problem data
	5	Print header SLICE 75
	6	Section limits ALPHA and BETA are in CPS
1	7	Calculate eigenvalue error bounds
	8	Print accepted subspace vectors
	9	Permit user changes in parameter values
1	10	Print iteration and shift history summary
1	11	Print iteration and shift history details
1	12	Use default subsection definition factor GAMMA
	13	Print coefficient matrix M ($S\underset{\sim}{x} = \text{lambda}*M\underset{\sim}{x}$)
	14	Print coefficient matrix S
	15	Print congruence matrix (SIMVEC,M*SIMVEC) before orthonormalization
	16	Print array name table
	17	Print table of spectral shift points
	18	Perform detailed trace of record operations
	19	Print set of initial vectors
	20	Print subspace vector iterates after each pass
	21	Print eigenvalue bound matrix after each pass
	22	Print orthogonalization eigenvalues
	23	Print name table of most of the arrays used internally
	24	Print the Chebyshev iterate acceleration coefficients
1	25	Modify limit ALPHA for improved efficiency when given ALPHA=0
	26-30	...Not presently used...

sophisticated studies of the algorithms used in SLICE 75. Further information in this respect is presented in Section 4. The casual user will normally only be interested in flags 1-7 for printout and perhaps 16 which prints the file table of contents after processing. Switch 18 produces extremely voluminous output and should not be enabled.

The next input request will be

```
ENTER THE EIGENVALUE LIMITS ALPHA AND BETA {IN CPS}
?
```

where the brackets indicate optional data depending upon the state of flag 6. The numbers entered at this time defines the section [ALPHA, BETA] of the eigenvalue spectrum on which eigenvalues will be determined. If ALPHA=0 and flag 25 is enabled, then ALPHA will be set to a negative value in order to improve the computational efficiency for eigenvalues in the vicinity of 0. This option is appropriate, for example, when analyzing structural problems which are non-negative definite and have important zero eigenvalues (rigid body modes).

2.2.3 Basic Input Summary and Examples

For a simple execution of SLICE 75, all the user input data required is as follows:

File ID
Group ID, Contents ID
Indexes of flags to be switched
ALPHA, BETA (section limits).

In Example 2.2 we show the results of an execution utilizing minimum user input.

Most of the printed output is reasonably self-explanatory. The subsection definition factor, subsequently called GAMMA, is discussed in Section 3. The order of iterate acceleration, also discussed in Section 3, is the degree of the Chebyshev polynomial used for iterate acceleration.

This example includes the iteration details (flag 10) which provides eigenvalue estimates as the iteration proceeds. These (dynamic) estimates are obtained by an acceleration process discussed in Section 3.4 and, at the time of acceptance, are more accurate than those displayed after subspace complete. Compare, for example, .44000000 and .44000005.

The number under CONV indicates how many of the 5 simultaneous iterates have converged after each iteration. Notice that even though both subspace vectors had converged after 3 accelerated iterations, the program continued for 6 more iterations in order to be sure the subspace was complete. After

```
          SLICE 75

 ENTER THE DATA BASE FILE ID
? 90753
 ENTER INPUT MATRIX GROUP AND CONTENT ID'S
? 10241, 4097

     PROGRAM CONTROL FLAGS    11110 01001    11000 00000    00001 00000

     ENTER INDEXES OF FLAGS TO BE SWITCHED ON ONE LINE
?

     ENTER THE EIGENVALUE LIMITS ALPHA AND BETA
? .28   .56

     PROBLEM SIZE ..................    35
     SECTION BOUNDS ................  2.80000000E-01  5.60000000E-01
     MAX. NUMBER OF SHIFT POINTS ...    14
     SUBSECTION DEFINITION FACTOR ..      .7000
     MIN. PERCENT OF SECTION COVERED    99.9991
     ORDER OF ITERATE ACCELERATION .     3
                    ---       1.655 - TIME AFTER PROBLEM SETUP
                    ---       1.737 - TIME AFTER SPECTRAL SHIFT SETUP
                    ---       1.837 - TIME AFTER SHIFT FACTORIZATION

                    --- SPECTRAL SHIFT  1 ---
               SHIFT POINT ..............  4.2000000E-01
               NO. OF SMALLER EIGVALS ....          0
               DIST. TO SECTION BOUNDARY .  1.4000000E-01
               CONVERGENCE ERROR CRITERION  2.3841860E-07

        INTERMEDIATE RELATIVE EIGENVALUE AND ERROR BOUND ESTIMATES

 CONV  ESTIMATE       ERROR       ESTIMATE       ERROR       ESTIMATE       ERROR
  0  4.4001711E-01  1.19E-04  4.4000157E-01  4.97E-06  4.4000847E-01  8.34E-06
     4.4027421E-01  1.51E-04  4.4851545E-01  1.68E-02
  1  4.4000000E-01  9.73E-09  4.4313953E-01  3.50E-01  5.7428404E-01  5.50E-01
     5.8694231E-01  4.02E-01  5.5524747E-01  9.98E-01
                    ---       3.601 - TIME AFTER ITERATION PASS
  1  5.9460123E-01  3.78E-01  5.0000000E-01  1.15E-07  5.8244491E-01  6.08E-01
     5.8613761E-01  3.37E-01  5.7898604E-01  3.97E-01
                    ---       4.687 - TIME AFTER ITERATION PASS
  0  6.2355864E-01  1.32E-01  5.9347817E-01  5.61E-01  5.9160775E-01  3.45E-01
     5.8381336E-01  3.29E-01  5.8834202E-01  2.62E-01
  0  6.0500105E-01  4.34E-01  5.9989586E-01  3.51E-01  5.9983758E-01  2.72E-01
     6.1820855E-01  1.62E-01  5.9473362E-01  1.79E-01
  0  6.2173230E-01  3.20E-01  6.1061196E-01  3.90E-01  5.9783769E-01  2.11E-01
     6.2455263E-01  1.98E-01  5.9308836E-01  3.32E-01
  0  6.6220098E-01  1.89E-01  6.2481250E-01  5.19E-01  6.0156906E-01  2.74E-01
     6.3528319E-01  5.94E-01  5.8729823E-01  5.87E-01
  0  7.1014218E-01  2.81E-02  6.4407713E-01  5.30E-01  5.9001093E-01  1.82E-01
     5.7645665E-01  6.29E-01  5.7870399E-01  8.01E-01
  0  6.7787250E-01  7.65E-03  6.3534972E-01  2.44E-01  5.7622737E-01  1.20E-01
     5.6962965E-01  6.29E-01  5.7300531E-01  8.72E-01
                    ---      11.368 - TIME AFTER ITERATION PASS

                    --- SUBSPACE COMPLETE ---
     THE ESTIMATED EIGENVALUES ARE
   1  4.40000005E-01  2  5.00000012E-01
                    ---      11.380 - TIME AFTER SUBSPACE COMPLETION
```

Example 2.2 Simplest Input Data

```
                    EIGENVALUES
                    -----------
 1  4.40000000E-01   2  5.00000023E-01

                    EIGENVALUES IN CPS
                    ------------------
 1  1.05571446E-01   2  1.12539542E-01

    MATRIX EIG. VECS.  (   30734)     SIZE.    8

    TYPE (16,1,0),   DIMENSIONS  25   2,   PRIORITY  9
    THE SUCCESSIVE VECTORS OF THE SET ARE
    VECTOR EIG. VECS.  (  133184)  TYPE ( 0,1,0), DIMENSION  25, PRIORITY 14
 1.98980E-01 -5.73861E-03 -3.64778E-03  2.57802E-03  5.89633E-04  6.28926E-03
 2.85673E-03  2.47562E-03 -2.40874E-03  1.49466E-03  4.67608E-03 -4.22720E-03
-1.05700E-03  5.11269E-03  2.84631E-03 -1.53424E-03  5.30532E-03 -6.24640E-03
-6.10590E-03 -2.46969E-03 -5.53434E-03  3.66734E-03  4.70472E-03 -4.61874E-03
-4.06225E-03
    VECTOR EIG. VECS.  (   49175)  TYPE ( 0,1,0), DIMENSION  25, PRIORITY  8
-5.85546E-03  1.71196E-01 -2.09294E-02  1.48327E-02  3.33772E-03  3.61083E-02
 1.64137E-02  1.42143E-02 -1.38147E-02  8.57961E-03  2.68342E-02 -2.42559E-02
-6.06597E-03  2.93357E-02  1.63314E-02 -8.80348E-03  3.04417E-02 -3.59414E-02
-3.50353E-02 -1.41709E-02 -3.17557E-02  2.10430E-02  2.69954E-02 -2.85020E-02
-2.33089E-02
                    ---      11.597 - TIME AFTER SOLUTION COMPLETE

                    MAXIMUM EIGENVALUE ERRORS
                    -------------------------
 1  1.73161618E-08   2  1.83940718E-05
                    ---      11.755 - TIME AFTER ERROR BOUNDS FOUND

        ------------------------------------------------
         THE SLICE75 DATA OUTPUT GROUP ID      12290
         THE CONTENTS RECORD ID                 6145
        ------------------------------------------------


        ------------------------------
         FILE ESTABLISHED ON   AUX1
         FILE ID              408385
        ------------------------------
                    ---      11.858 - TIME AFTER JOB FINISHED
   11.868 CP SECONDS EXECUTION TIME
```

Example 2.2 (Continued)

all the subspace vectors have been found, SLICE 75 continues until the maximum "age" (number of iterations applied to a vector) of a simultaneous iterate reaches a specified limit. When many eigenvalues are calculated, the final iteration to ascertain completeness represents a relatively small overhead.

The eigenvalues finally printed out result from the final resolution of subspace vectors into eigenvectors. This stage of the processing is further discussed in Section 3. For Example 2.4, below, we ran this same problem with a tighter acceptance tolerance to obtain full accuracy in the eigenvalues.

The eigenvector printout is in the standard format for matrices in the form of vector sets as handled by the matrix-vector utilities, see Appendix B. The maximum eigenvalue errors are conservative bounds on the error in each eigenvalue obtained as discussed in Section 3. If one of these bounds is very small, the user can be confident that his solution is very accurate. If, on the other hand, it is large, the solution may still be very good, i.e., the bound may be grossly over-conservative.

In Example 2.3 we show results from a slightly more complex set of input data. Flag 3 was switched twice to show the effect of repeated switching. Notice that a comma, spaces or both can be used to delimit free field input. Also integers can be entered as reals and vice-versa.

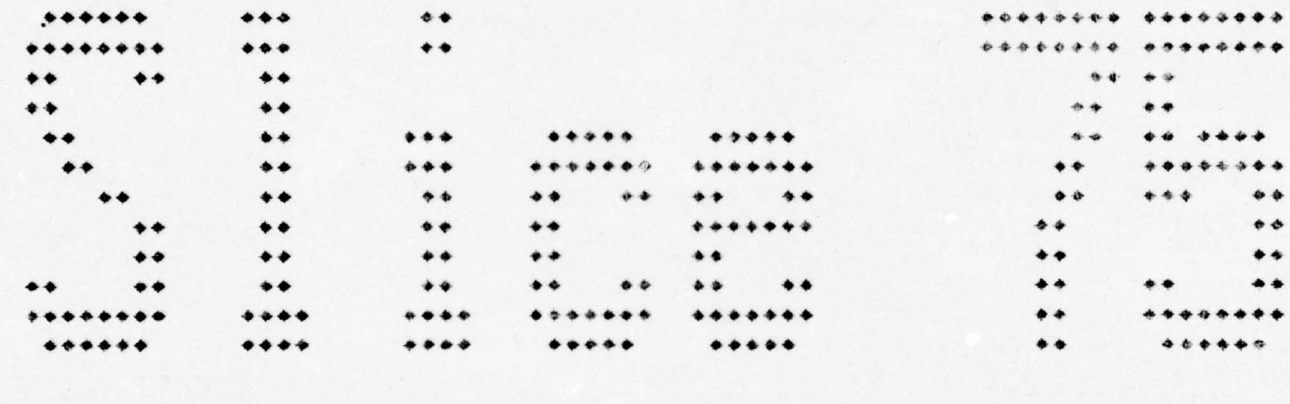
2.3 Some Operational Considerations

The efficiency of an eigenvalue analysis of a certain section of the spectrum can be dramatically influenced by the choice of ALPHA, BETA and, to a lesser extent, by GAMMA (flag 12). Thus, it is possible for a user to utilize knowledge he may have about the eigenvalues in order to reduce the cost of the analysis.

2.3.1 Choosing ALPHA and BETA

In Figure 2.4 we show two choices for limits ALPHA and BETA on a given spectrum. Although both define the same set of eigenvalues, the solution time using (ALPHA2, BETA2) is less than 70% of that for (ALPHA1, BETA1). If the

```
          SLICE 75

 ENTER THE DATA BASE FILE ID
? 90753
 ENTER INPUT MATRIX GROUP AND CONTENT ID'S
? 10241  4097

     PROGRAM CONTROL FLAGS    11110 01001    11000 00000    00001 00000

     ENTER INDEXES OF FLAGS TO BE SWITCHED ON ONE LINE
? 2 3 5 16 11 12 3

     PROGRAM CONTROL FLAGS    10111 01001    00000 10000    00001 00000

     ENTER THE EIGENVALUE LIMITS ALPHA AND BETA
? .3447, .98

     THE DEFAULT SUBSECTION DEFINITION FACTOR GAMMA IS  .7000000
     ENTER THE VALUE DESIRED
? .71
1
```



```
     PROBLEM SIZE ..................    25
     SECTION BOUNDS ................   3.447000000E-01  9.800000000E-01
     MAX. NUMBER OF SHIFT POINTS ...    13
     SUBSECTION DEFINITION FACTOR ..      .7100
     MIN. PERCENT OF SECTION COVERED   99.9993
     ORDER OF ITERATE ACCELERATION .     3
                    ---       1.735 - TIME AFTER PROBLEM SETUP
                    ---       1.915 - TIME AFTER SPECTRAL SHIFT SETUP
                    ---       1.918 - TIME AFTER SHIFT FACTORIZATION

                    --- SPECTRAL SHIFT  1 ---
               SHIFT POINT ..............  6.623500E-01
               NO. OF SMALLER EIGVALS ....          4
               DIST. TO SECTION BOUNDARY .  3.178500E-01
               CONVERGENCE ERROR CRITERION  2.384186E-07
                    ---       3.751 - TIME AFTER ITERATION PASS
                    ---       4.839 - TIME AFTER ITERATION PASS
                    ---       6.207 - TIME AFTER ITERATION PASS
                    ---       8.816 - TIME AFTER ITERATION PASS
                    ---      10.257 - TIME AFTER ITERATION PASS
                    ---      11.815 - TIME AFTER ITERATION PASS
                    ---      18.045 - TIME AFTER ITERATION PASS
```

Example 2.3 Changing Control Switches and Subsection Parameter

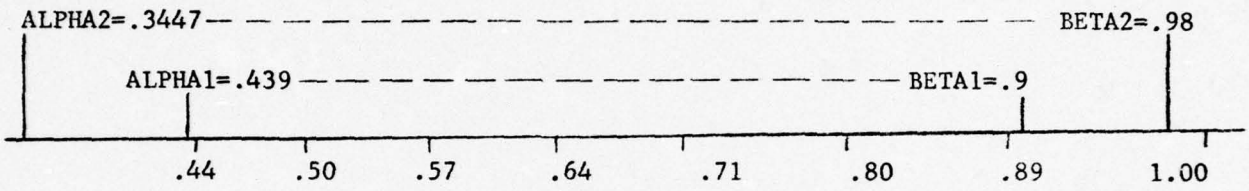
```
                    --- SUBSPACE COMPLETE ---
  THE ESTIMATED EIGENVALUES ARE
 1  6.36363620E-01  2  7.14285714E-01  3  5.65217303E-01  4  8.00000004E-01
 5  4.99999973E-01  6  8.94736911E-01  7  4.39999997E-01
                    ---      18.063 - TIME AFTER SUBSPACE COMPLETION

                    EIGENVALUES
                    -----------
1  4.40000010E-01  2  5.00000000E-01  3  5.65217391E-01  4  6.36363636E-01
5  7.14285714E-01  6  8.00000000E-01  7  8.94736917E-01

                    EIGENVALUES IN CPS
                    ------------------
1  1.05571447E-01  2  1.12539540E-01  3  1.19654184E-01  4  1.26961723E-01
5  1.34510477E-01  6  1.42352509E-01  7  1.50545511E-01
                    ---      18.785 - TIME AFTER SOLUTION COMPLETE

                    MAXIMUM EIGENVALUE ERRORS
                    -------------------------
1  2.80558777E-05  2  1.45739101E-06  3  1.04066990E-07  4  7.29699968E-10
5  4.60910735E-11  6  4.36488250E-07  7  5.87616646E-05
                    ---      19.413 - TIME AFTER ERROR BOUNDS FOUND

   THE  8 PRIMARY RECORD ID'S ARE

        RECORD       ID
        ----------------
        CNTNTS      6146
        PARAM       4097
        S          12293
        M           8195
        FSM        38930
        EIGVL      14342
        BOUNDS     34832
        EIGVEC     30734
```

Example 2.3 (continued)

user has the option, he should adopt the principle: Choose ALPHA and BETA to maximize the quantity

$$\min_{\lambda \in \mathcal{A}} (\text{BETA} - \lambda , \ \lambda - \text{ALPHA})$$

where $\mathcal{A}$ is the set of all eigenvalues such that ALPHA $< \lambda <$ BETA.



Solution times:

(ALPHA1, BETA1) - 27.4
(ALPHA2, BETA2) - 18.8

Fig. 2.1 Two Choices of ALPHA and BETA

2.3.2 Large Sections

Practice has shown that it is generally beneficial to break up large sections (ranges (ALPHA, BETA) containing more than 40 eigenvalues) into smaller ones and carrying out several analyses. Sections containing 15 to 25 eigenvalues are generally handled very well. The default capacity limitation (specified by parameter MAXEV, see Table 4.3) is 50.

Occasionally, sophisticated users attempt the analysis of a large section with a large set of simultaneous iterates, but allow insufficient time for completion of the analysis. They then use the eigenvalue estimates obtained for constructing a clever subdivision of the section by means of the principle given in Section 2.3.1 as illustrated in Figure 2.5. Such "games" are only played by people solving very large problems who are very concerned about

computer costs. They are neither required nor advisable for normal analysis. Furthermore, several minor extensions to SLICE 75, such as the ability to input selected initial vectors, are required for the proper playing of the efficiency games.

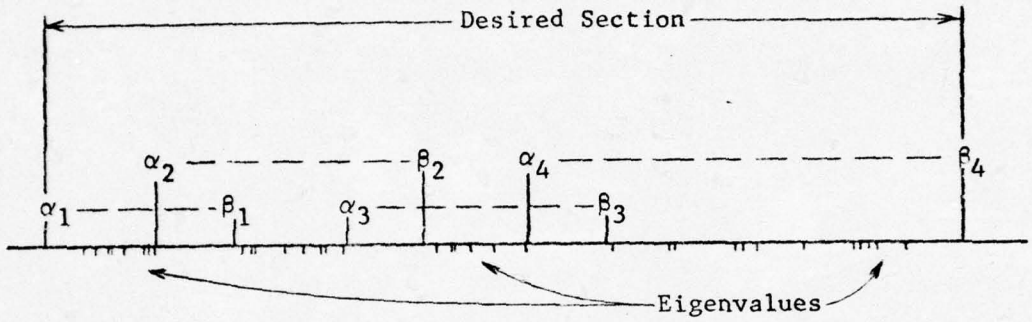


Fig. 2.2 Clever Subdivision of a Large Section

2.3.3 Refined Solutions

Occasionally there are certain eigenvectors which are wanted to extremely high accuracy. There are two approaches to achieving this objective, either or both of which may be invoked:

1. One can choose ALPHA and BETA close to each other so that the desired eigenvalue

$$\lambda \approx (\text{ALPHA} + \text{BETA})/2$$

is isolated, or nearly so. Note that unless pivoting is used in the factorization process (it usually is not), equality in the above expression is undesirable.

2. One can decrease the acceptance criterion (parameter TOLMCH, see Table 4.3).

Both options have their associated costs: 1) in an additional factorization and 2) in more iterations. The choice is a matter of experience and judgment. Example 2.4 illustrates the mechanics of reducing the acceptance tolerance TOLMCH.

```
          SLICE 75

 ENTER THE DATA BASE FILE ID
? 90753
 ENTER INPUT MATRIX GROUP AND CONTENT ID'S
? 10241, 4097

     PROGRAM CONTROL FLAGS    11110 01001    11000 00000    00001 00000

     ENTER INDEXES OF FLAGS TO BE SWITCHED ON ONE LINE
? 2, 9

     PROGRAM CONTROL FLAGS    10110 01011    11000 00000    00001 00000

     ENTER THE EIGENVALUE LIMITS ALPHA AND BETA
? .29, .55

     ENTER NO. OF PARAMETER CHANGES TO BE MADE AND THE WORD  DISPLAY
     IF THE CURRENT PARAMETERS ARE TO BE DISPLAYED.
? 1

     ENTER CHANGES - INDEX,VALUE,INDEX,VALUE,ETC.
? 15,1.E-9
     CHANGE PARAMETER 15 FROM  2.3841858E-07 TO  1.0000000E-09

     PROBLEM SIZE ..................    25
     SECTION BOUNDS ................  2.30000000E-01  5.50000000E-01
     MAX. NUMBER OF SHIFT POINTS ...    14
     SUBSECTION DEFINITION FACTOR ..      .7000
     MIN. PERCENT OF SECTION COVERED   98.9991
     ORDER OF ITERATE ACCELERATION .     3
                    ---       1.671 - TIME AFTER PROBLEM SETUP
                    ---       1.756 - TIME AFTER SPECTRAL SHIFT SETUP
                    ---       1.857 - TIME AFTER SHIFT FACTORIZATION

                    --- SPECTRAL SHIFT  1 ---
               SHIFT POINT ..............  4.2000000E-01
               NO. OF SMALLER EIGVALS ....          0
               DIST. TO SECTION BOUNDARY .  1.3000000E-01
               CONVERGENCE ERROR CRITERION  1.0000000E-09

          INTERMEDIATE RELATIVE EIGENVALUE AND ERROR BOUND ESTIMATES

 CONV  ESTIMATE       ERROR      ESTIMATE       ERROR      ESTIMATE       ERROR
  0  4.4001711E-01  1.38E-04  4.4000157E-01  5.78E-06  4.4000847E-01  9.70E-06
     4.4027421E-01  1.76E-04  4.4851545E-01  1.97E-02
  0  4.4000000E-01  1.02E-08  4.4574370E-01  3.08E-01  5.9837562E-01  4.57E-01
     6.1506765E-01  3.30E-01  5.1861905E-01  9.83E-01
  1  4.4000000E-01  6.31E-16  5.0000001E-01  4.5AE-07  6.3740302E-01  1.38E-01
     6.4010064E-01  2.06E-01  6.5210427E-01  3.21E-01
                    ---       4.449 - TIME AFTER ITERATION PASS
  0  6.7578509E-01  2.69E-01  5.0000000E-01  1.99E-09  6.3800354E-01  1.51E-01
     6.3937660E-01  1.83E-01  6.5758715E-01  2.47E-01
  1  5.0000000E-01  7.54E-12  6.3796947E-01  1.78E-01  6.3861973E-01  1.49E-01
     6.5451265E-01  2.22E-01  6.7523357E-01  2.33E-01
                    ---       6.536 - TIME AFTER ITERATION PASS
```

Example 2.4 Decreasing the Acceptance Tolerance, see Table 4.3

```
0  6.3967330E-01   1.43E-01   6.3840922E-01   2.16E-01   6.3805668E-01   9.62E-02
   6.5163434E-01   2.61E-01   6.7690219E-01   3.17E-01
0  6.4024054E-01   2.55E-01   6.3735542E-01   3.80E-02   6.5013534E-01   4.14E-01
   6.6285729E-01   3.18E-01   6.6621213E-01   3.80E-01
0  6.4882764E-01   2.48E-01   6.3660890E-01   3.57E-03   6.5841161E-01   4.52E-01
   6.8162011E-01   5.81E-01   7.6638183E-01   5.85E-01
0  6.8460647E-01   8.65E-02   6.3638910E-01   2.87E-04   7.1161943E-01   1.71E-01
  -7.7109222E-01   4.13E-01   8.9448767E-01   3.38E-01
                   ---      10.998 - TIME AFTER ITERATION PASS

                   --- SUBSPACE COMPLETE ---
   THE ESTIMATED EIGENVALUES ARE
 1  4.40000000E-01  2  5.00000000E-01
                   ---      11.008 - TIME AFTER SUBSPACE COMPLETION

                   EIGENVALUES
                   -----------
1  4.40000000E-01  2  5.00000000E-01

                   EIGENVALUES IN CPS
                   ------------------
1  1.05571446E-01  2  1.18539540E-01
                   ---      11.198 - TIME AFTER SOLUTION COMPLETE

                   MAXIMUM EIGENVALUE ERRORS
                   -------------------------
1  2.26302468E-11  2  1.89831169E-07
                   ---      11.372 - TIME AFTER ERROR BOUNDS FOUND
```

Example 2.4 (continued)

This example shows the solution of the same problem as Example 2.1. Note that even though 5 accelerated iterations (compared to 3 for Example 2.1) were required for convergence of the two subspace vectors, the total number of iterations taken was the same as for Example 2.1. This, of course, is due to the subspace completion overhead discussed earlier. Note that had convergence required 9 iterations, there would have been no overhead and the total cost would still have been the same.

Section 3
EXTENDED SECTIONING METHOD

The sectioning eigensolution technique for the large, symmetric, generalized problem (1), where M is positive definite was introduced by Jensen [4]. The algorithm is designed to calculate the eigenvalues in a given interval of the eigenvalue spectrum and the corresponding eigenvectors. Generally, the number of eigenvalues in the interval is small compared to the order of the system.

Inverse iteration is used to determine a set of vectors which span the subspace (section) defined by the desired eigenvectors rather than attempting to determine the eigenvectors directly. This results in a considerable reduction in computational cost and simplifies the difficulties associated with clustered eigenvalues. In fact, in most cases the presence of multiple or clustered eigenvalues tends to improve the behavior of the algorithm.

A currently popular improvement to simple inverse iteration is simultaneous iteration which has been discussed by Rutishauser [8], Stewart [9], Bathe [1], McCormick [7] and others. The effect of simultaneous iteration is similar to that of sectioning in the respect that convergence to a subspace instead of to a vector is all that is required. For very large systems, however, iteration on several vectors simultaneously also reduces the total transfer of data between mass storage and central memory. For this reason it is beneficial to incorporate simultaneous iteration in the sectioning algorithm.

Another improvement to inverse iteration is obtained by the use of Chebyshev polynomials to accelerate the iterates as briefly discussed, for example, in Householder [3] and Rutishauser [8]. The use of Chebyshev acceleration turns out to be particularly atttractive with the sectioning algorithm because of the fact that it operates on a specified interval of the eigenvalue spectrum.

In the next subsection we shall briefly describe the sectioning algorithm with iterate acceleration and simultaneous iteration. In the following subsection, we discuss the iterate acceleration process in some detail and show some numerical results.

3.1 Extended Sectioning Algorithm

Assume that eigenvalues of (1) in the open interval (α, β) and the corresponding eigenvectors are to be calculated. Denote those eigenvalues by $\lambda_1, \ldots, \lambda_m$ and the corresponding eigenvectors by $\underset{\sim}{x}_1, \ldots, \underset{\sim}{x}_m$. Let the "section" H be the subspace which is spanned by $\underset{\sim}{x}_1, \ldots, \underset{\sim}{x}_m$. We seek to determine a set of vectors $Y = [\underset{\sim}{y}_1, \ldots, \underset{\sim}{y}_m]$ which spans H.

Let the "control" parameter γ $(0 < \gamma < 1)$ be given. The meaning and purpose of γ will become apparent subsequently. Define a sequence of "range values" $\{\sigma_i\}$ and "shift points" $\{\mu_i\}$ by

$$\sigma_i = \left(\frac{1-\gamma}{1+\gamma}\right)^i (\beta-\alpha)/2, \tag{2}$$

and

$$\begin{aligned} \mu_{2i} &= \alpha + \sigma_i \\ \mu_{2i-1} &= \beta - \sigma_i, \quad i = 0, 1, \ldots, n_\sigma, \end{aligned} \tag{3}$$

where n_σ is discussed below. Thus, for each shift point μ, the corresponding range value σ is given by

$$\sigma(\mu) = \min\ (\beta-\mu, \mu-\alpha)\ . \tag{4}$$

With each shift point μ we associate a "subinterval"

$$\xi(\mu) = (\mu - \gamma\sigma(\mu), \quad \mu + \gamma\sigma(\mu))\ ,$$

which is used by the sectioning algorithm to determine when the shift point is

to be changed. Notice that the fraction of the interval (α, β) which is contained in the union of the subontervals $\xi_i = \xi(\mu_i)$, called the "coverage", is given by

$$\text{cov} = 1 - (1-\gamma)\left(\frac{1-\gamma}{1+\gamma}\right)^{n_\sigma} \quad . \tag{5}$$

Normally the control parameter γ and the desired minimum coverage are specified and the appropriate value n_σ is then computed from (5). For example, if $\gamma = .7$ and $\text{cov} \geq .999$, then $n_\sigma = 4$. The coverage designates the portion of the given interval over which eigenvalues are most certain to be found. The control parameter γ determines the maximum number of spectral shifts $2n_\sigma + 1$ that may be required to achieve a specified coverage. As γ increases, n_σ decreases but the maximum number of iteration steps per shift generally increases. A suitable approach for choosing γ is presented in Section 3.2.3.

3.1.1 Algorithm

In order to put the processes involved into perspective, we now present a rough description of the extended sectioning algorithm. The details are then discussed subsequently.

For given coefficients $c_o, c_1, \ldots c_k$, let polynomial

$$p_k(x) = \sum_{j=0}^{k} c_j x^j \tag{6}$$

and an array of M-orthonormal initial vectors

$$V^{(0)} = [\underset{\sim}{v}_1^{(0)}, \ldots, \underset{\sim}{v}_s^{(0)}] \tag{7}$$

be given. As subspace vectors are determined, they will be appended to an array Y which, of course, is initially empty. Then the extended sectioning algorithm may be roughly described as follows:

1. Spectral shift. The first shift point is always $\mu = \mu_o$. Then shift points with odd index are taken in order of increasing index followed by those with even index in the same order. Under some circumstances, those with even index may be taken first.

   set μ to the next shift point

   $\sigma := \sigma(\mu)$

   $i := 0$

2. Krylov sequence.

$$A := (I - Y Y^T M)\,(S - \mu M)^{-1} M$$

$$Z^{(0)} := V^{(i)}$$

$$Z^{(j)} := A \cdot Z^{(j-1)}, \qquad j = 1, \ldots, k$$

3. Iterate acceleration.

$$W := \sum_{j=0}^{k} c_j \sigma^j Z^{(j)}$$

4. Orthonormalization.

$$T := W^T M W$$

   Solve small s by s eigenproblem

$$TQ = QD^2, \quad Q\text{-unitary, } D\text{-diagonal}$$

$$W := WQD^{-1}$$

5. Next approximation.

$$i := i + 1$$

$$V^{(i)} := W$$

6. Acceptance test.

   Calculate eigenvalue bounds b_i, $i = 1, \ldots, s$ corresponding to the vector iterates.

6. Acceptance test (Continued)

Apply convergence test. If any vectors pass, go to 7.

If $\min_i (b_i) \le \gamma \sigma$, go to 2.

If $i < i_{max}$, go to 2.

If $\min_i (b_i) \le \sigma$, go to 1.

If both odd and even sets of shift points have been used, go to 8.

Switch shift point selection to set with opposite (odd-even) type go to 1.

7. Vector acceptance.

Append to Y all converged vectors in $V^{(i)}$.

Replace converged vectors in $V^{(i)}$ with new initial vectors.

If any estimated eigenvalue corresponding to a non-converged vector in $V^{(i)}$ lies in the interval $(\mu-\gamma\sigma, \mu+\gamma\sigma)$ go to 2.

If the "ages" of the vectors in $V^{(i)}$ are all less than MAXAGE, go to 2.

If not all shift points have been used, go to 1.

8. Resolution.

$T := Y^T SY$

Solve small eigenproblem $TQ = Q\Lambda$ where Λ is diagonal and Q is unitary.

Calculate eigenvectors $X := YQ$.

9. Calculate *a-posteriori* error bounds.

3.1.2 General Comments.

Notice that in step 4, the accelerated iterates W are M-orthonormal and orthogonal to previously accepted subspace vectors Y. This is required to

make the final resolution step 8 correct. Notice also that a unitary-diagonal QD^{-1} orthonormalization matrix is used instead of the more common upper triangular matrix such as appears in the Gram-Schmidt process. The Gram-Schmidt technique generally yields slower convergence because it tends to force the first column of V to converge toward the eigenvector corresponding to the eigenvalue nearest the shift point, the second column to the eigenvector with eigenvalue second nearest the shift point, and so forth. Consequently, if a column of V is initially "weak" in the vector toward which it is forced to converge, excessive iteration is required. This unpleasant condition can be improved somewhat by rearranging the iterates after each orthogonalization. However, the solution of the s by s eigenproblem by the Jacobi method tends to be very inexpensive since T becomes increasingly diagonally dominant.

Although the matrix $V^{(i)}$ of step 2 is orthonormal, it is possible for the columns of W in step 3 to be very nearly linearly dependent. Consequently, after determining the matrix D in step 4, the vectors of the final W corresponding to very small values of d_i are discarded.

Since converged vectors in V are regularly being replaced by new "initial" vectors, it is necessary to keep track of the "age" of each vector in V. In this context, the age of a vector is the number of iterations performed since its introduction in V. When the age of the oldest vector in V reaches a threshold MAXAGE (see step 7 of the algorithm), processing at the current shift point is terminated.

3.2 Iterate Acceleration

In step 3 of the sectioning algorithm described above, we indicate that an accelerated convergence of the iterates Z can be achieved by forming a linear combination of previously determined iterates. In this section we shall show how that is achieved and some quantitative estimates of the benefits which result.

3.2.1 Theoretical Considerations

Let n by n matrix X be the complete array of M-orthogonal eigenvectors of (1), and let Λ be the diagonal matrix of corresponding eigenvalues. Assume

for the moment $Y = 0$ and μ is not an eigenvalue. Then there exists an n by s "coefficient" matrix A such that

$$Z^{(j)} = X\,(\Lambda-\mu I)^{-j} A, \quad j = 0,1, \ldots , \tag{8}$$

where the iterates $Z^{(j)}$ are formed in step 2 of the sectioning algorithm. Consequently, we have from step 3

$$W = \sum_{j=0}^{k} c_j\, \sigma^j\, Z^j$$

$$= X \sum_{j=0}^{k} c_j\, \sigma^j\, (\Lambda-\mu I)^{-j} \quad A ,$$

or from (6)

$$W = X\, p_k\, (G)\, A, \tag{9}$$

where, letting $g_i = \sigma / (\lambda_i-\mu)$, $i = 1, \ldots, n$, we have $G = \mathrm{diag}\,(g_i)$.

As indicated earlier, the objective of the inverse iteration (step 2) is to eliminate the "contamination" of the iterates, i.e., eigenvector components corresponding to eigenvalues λ such that $|\lambda-\mu| \geq \sigma$. Stated another way, we wish to choose a polynomial $p_k(g)$ which will have a minimum, maximum value over the range $-1 \leq g \leq 1$. It is well known that this criterion is satisfied by the k^{th} order Chebyshev polynomial defined recursively by:

$$\begin{aligned} p_o(g) &= 1 \\ p_1(g) &= g \\ p_k(g) &= 2g\, p_{k-1}(g) - p_{k-2}(g), \quad k = 2, \ldots . \end{aligned} \tag{10}$$

3.2.2 Benefits

Since

$$\max_{|g| \le 1} |p_k(g)| = 1, \quad k = 0, 1, \ldots ,$$

we obtain a measure of the improvement per step in a vector with a component corresponding to eigenvalue $|\lambda - \mu| < \sigma$ from the expression

$$R_k(r) = \left| p_k\left(\frac{1}{r}\right) \right|^{-\frac{1}{k}} , \tag{11}$$

where $r = (\lambda - \mu)/\sigma = 1/g.$

Thus, $R_k(r)$ is the relative reduction in the size of the largest "contamination" component of the vector iterate per iteration step, using k^{th} order Chebyshev iterate acceleration, where r is the magnitude of the smallest eigenvalue relative to the range factor σ. The function $R_k(r)$ is illustrated in Fig. 3.1 for $k = 1, \ldots, 5$. We notice that the most dramatic improvement is from $k = 1$, conventional iteration, to $k = 2$. The realized benefit in going to higher order acceleration formulas decreases with the order.

This is an important observation because as the order gets higher, the number of required iterations between orthonormalization steps increases. That, in turn, leads to a loss of independence of the vectors, i.e., all of the iterates start converging to the same vector. Consequently, there is a trade-off in going to higher orders of acceleration in simultaneous iteration.

3.2.3 Some Cost Considerations of the Algorithm

The actual behavior of the sectioning algorithm for a given control parameter is very problem dependent. For example, in many cases only one or two shift points are used even though the possibility for using more must be built into the system. Consequently, our study of the behavior of the algorithm must be based on a type of problem which we can characterize but which may not ever actually occur.

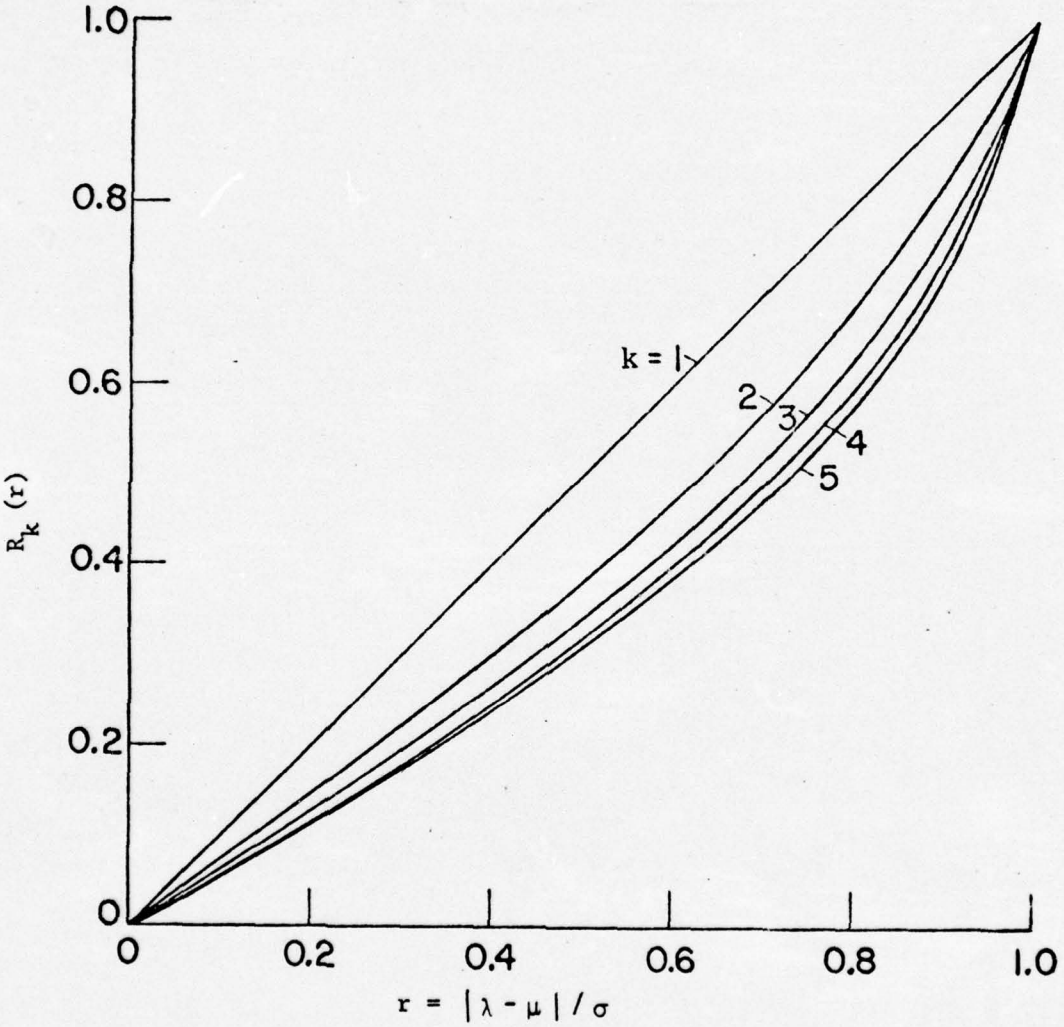


Fig. 3.1 Relative reduction in the size of the largest "contamination" component of a vector iterate per iteration step using k^{th} order Chebyshev acceleration.

The worst possible case is easily characterized by the clustering of eigenvalues at the extremes of such subinterval $\xi(\mu)$ for all shift points μ given by (3). Notice then that the problem must vary with the control paramater γ, which would make a meaningful numerical study with varying γ difficult, if not impossible. Nevertheless we shall use this case for the analysis of the algorithm.

Consider the partition

$$X = [X_s, X_c] \tag{12}$$

of the eigenvector matrix such that X_s spans the desired section H corresponding to interval (α, β), see Section 3.3.1. As in (8), let the initial vector iterates be given by

$$\begin{aligned} Z^{(0)} &= XA \\ &= X_s A_s + X_c A_c \ . \end{aligned} \tag{13}$$

Assume that multiplication of the comtimination component $X_c A_c$ in (13) by a factor ϵ removes it from $Z^{(0)}$ to machine accuracy, i.e., for some appropriate norm

$$f_\ell \left(\| A_s \| + \epsilon \| A_c \| \right) = \| A_s \|$$

where $f_\ell(a)$ is a computer floating point operator.

Then from (11), the number of iteration steps required to eliminate the contamination component is given by

$$n_k = k \left[-\log(\epsilon) / \log(p_k(1/\gamma)) \right], \tag{14}$$

where $[a]$ is the smallest integer exceeding $a > 0$, using control parameter γ for r. The functions $n_k(\gamma)$ are shown in Fig. 3.2 for $\epsilon = 10^{-8}$. Again we note the most dramatic improvement in going from order 1 to order 2 Chebyshev iterate acceleration.

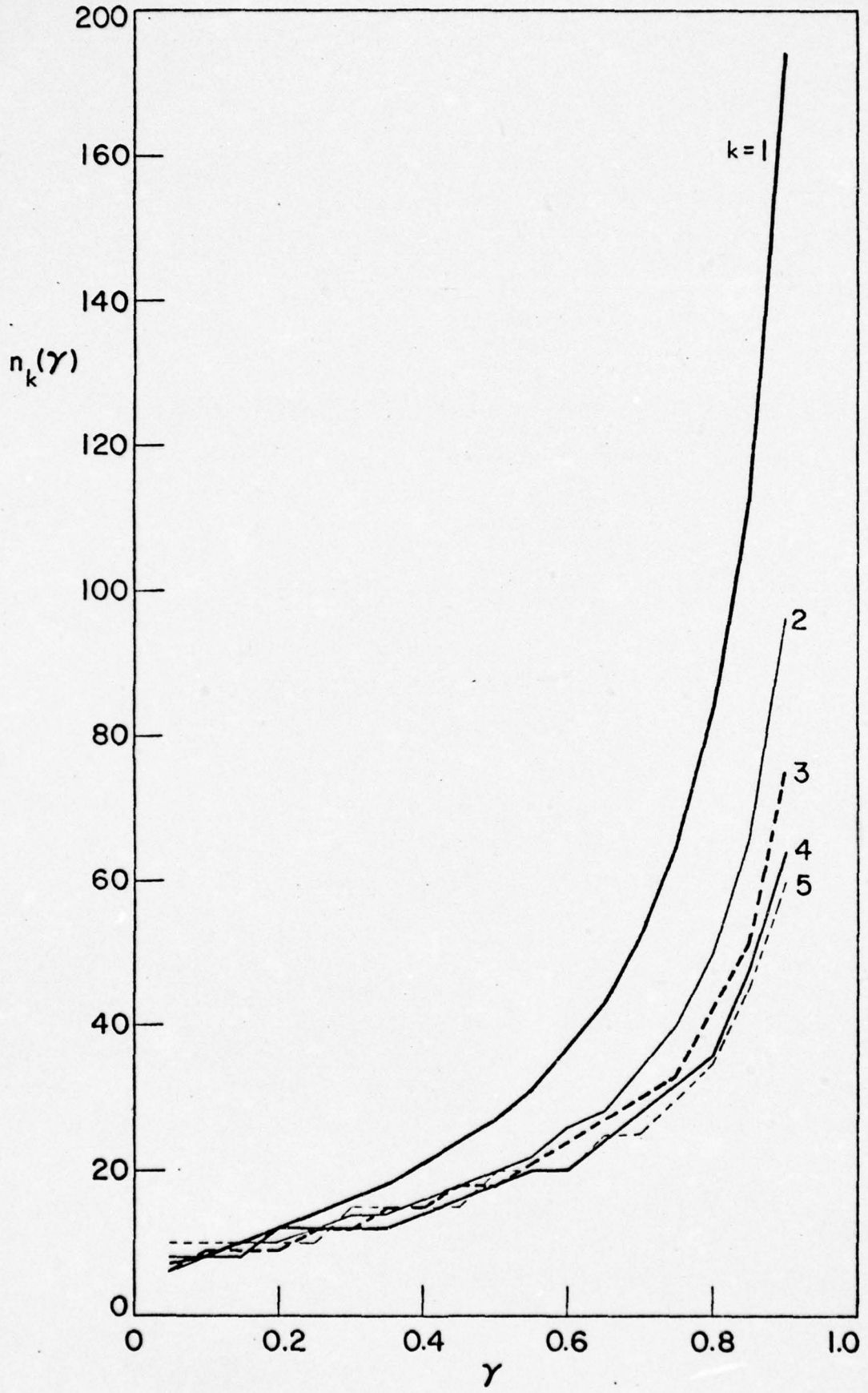


Fig.3.2. Maximum number of iterations required to reduce the contamination by a factor of $\epsilon = 10^{-8}$, using order k acceleration

As noted in Section 3.1, the control parameter γ regulates the trade-off between spectral shifting and iteration. Suppose the computational effort involved in one spectral shift is equivalent to that of q iteration steps as described in Item 3 of the sectioning algorithm. Then a normalized total cost in terms of the control parameter γ for solving the worst case eigenproblem is roughly characterized by

$$\text{cost } (\gamma) = n_s(\gamma)\ (n_i(\gamma) + q) \tag{15}$$

where the number of iteration per shift n_i is obtained from (14) and the number of shifts $n_s = 2n_\sigma + 1$ is obtained from (5). The cost function (15) is shown in Fig. 3.3 for two values of q. It is interesting to note that a value of about .72 for the control parameter γ yields a minimal total cost for nearly all of the illustrated cases.

As promised in Section 3.1, we have thus obtained a rationale for the selection of γ by analysis of a "worse case" problem which will probably never be encountered in practice. Experience with the program on practical problems will either verify this selection or lead to an improved rationale.

3.3 Vector Acceptance

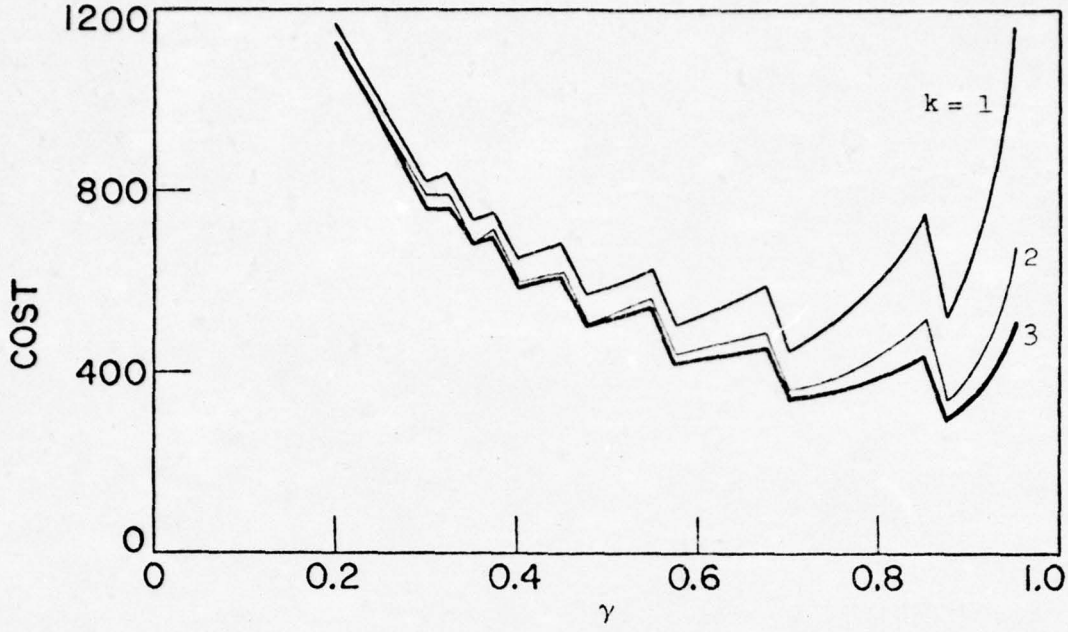
Referring to step 2 of the algorithm in Section 2.1.1, define s by s matrices

$$B^{(j)} = Z^{(j)^T} M Z^{(j)}, \quad j = 0, \ldots, k \tag{16}$$

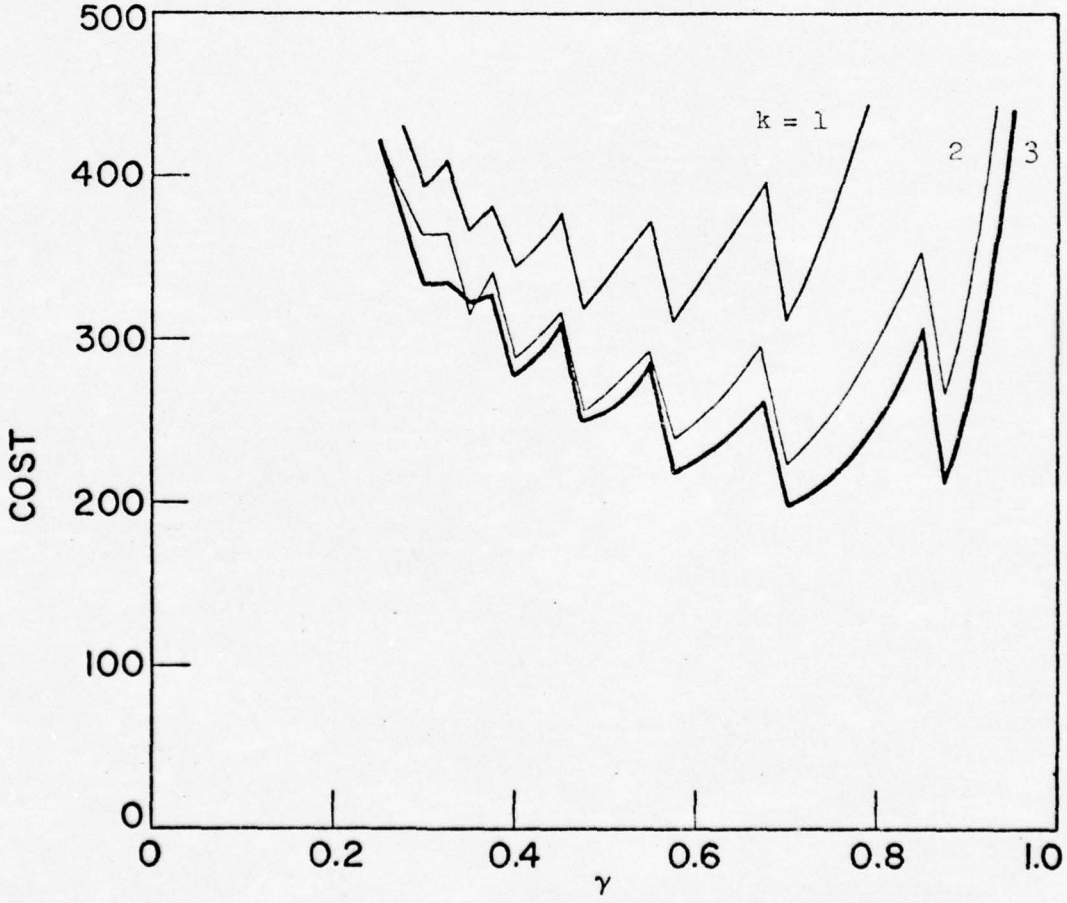
and vectors $\underset{\sim}{b}_j$, $j = 1, \ldots, k$, such that the i^{th} component b_{ij} of $\underset{\sim}{b}_j$ is determined from diagonal elements of $B^{(j-1)}$ and $B^{(j)}$ by

$$b_{ij}^2 = B_{ii}^{(j-1)}/B_{ii}^{(j)}, \quad i = 1, \ldots, s. \tag{17}$$

Then, as discussed in [1], there exists at least one eigenvalue λ of (1) such that



One factorization costs 40 iteration passes



One factorization costs 10 iteration passes

Fig. 3.3 Total cost function for worst case problem using k order acceleration and an error reduction factor of $\varepsilon = 10^{-8}$

(18) $$|\lambda - \mu| \leq b_{ij}$$

for all $i = 1, \ldots, s$, $j = 1, \ldots, k$. Each bound b_{ij} is an estimate obtained entirely from the i^{th} components of $Z^{(j-1)}$ and $Z^{(j)}$, and converges monotonically with j to a limit. The orthonormalization of step 4 prevents the convergence of these bounds to the same limit, except when a multiple eigenvalue exists.

The details of the acceptance technique which uses the bounds b_{ij} are covered in Section 4 of [4]. Briefly, we let $\underset{\sim}{z}_i^{(j)}$ by the i^{th} column of $Z^{(j)}$ and consider the form

(19) $$\underset{\sim}{z}_i^{(j)} = \underset{\sim}{u}_i^{(j)} + \underset{\sim}{v}_i^{(j)}$$

where $\underset{\sim}{u}_i^{(j)}$ is the projection of $\underset{\sim}{z}_i^{(j)}$ into the invariant subspace (section) H. Letting

(20) $$\psi_i^{(j)} = \| \underset{\sim}{u}_i^{(j-1)} \|^2 / \| \underset{\sim}{u}_i^{(j)} \|^2$$

and

(21) $$\delta_i^{(j)} = \| \underset{\sim}{v}_i^{(j)} \|^2 / \| \underset{\sim}{u}^{(j)} \|^2$$

we have

Theorem.

(22) $$\delta_i^{(j-1)} \leq (b_{ij}^2 - h_i^2 \psi_i^{(j)}) / (\sigma^2 - b_{ij}^2), \quad j = 2, \ldots$$

where $$h_i = \min_{\lambda \in \Lambda_i} |\lambda - \mu| = \lim_{j \to \infty} b_i^{(j)}$$

with Λ_i being the set of all eigenvalues

of (1) exclusive of

(i) those corresponding to accepted vectors

(ii) those correspondging to columns $z_{\sim k}^{(j)}$, $k = 1, \ldots, i-1$.

The missing ingredient in eq. (22) is an estimate for $h_i^2 \psi_i^{(j)}$. This is taken to be the estimate $\overline{b_{ij}}^2$ of the limit of the sequence $b_{i,1}$, $b_{i,2}$, ..., b_{ij} obtained by the acceleration procedure discussed in [5]. It is the smaller (real) root of the quadratic

$$a\, x^2 - b\, x + c$$

where

$$a = (b_{i,j-1}^2 - b_{ij}^2) / b_{i,j-1}^2$$

$$b = b_{i,j-2}^2 - b_{ij}^2$$

$$c = (b_{i,j-2}^2 - b_{i,j-1}^2)\, b_{ij}^2 .$$

Thus, the vector iterate $z_i^{(j)}$ is accepted when the right side of (22) is less than a prescribed bound, parameter TOLMCH in the program.

3.4 Error Bounds

A simple technique for obtaining rigorous a posteriori error bounds is discussed in Section 6 of [4]. Unfortunately, however, a factorization of either S or M is required, which for general large problems is prohibitively expensive.

In SLICE 75 we have immediately available a factorization of $(S - \mu M)$ for the last spectral shift point μ. Consequently, we use a slight extension of the technique of [4] which uses the available factorization. In practice, the bounds obtained turn out to be not as sharp as the more costly ones, particularly when μ is close to an eigenvalue. Thus, when a small bound is obtained, the user can be confident that he indeed has a good solution.

Section 4

PROGRAM DOCUMENTATION

As mentioned in the introduction of this manual, an important consideration in the design of SLICE 75 is independence of matrix size and format. Compliance with this consideration has been achieved by making SLICE 75 an executive processor supported by a general purpose data manager and a collection of utility programs.

All vectors, matrices and other data are maintained as records or groups of records in a general storage pool by the storage manager. The detailed operation of the storage manager AID appears in [6]. A reasonably complete summary of its operation is provided in Appendix A in order to make this manual self-contained.

The utilities supporting SLICE 75 are of two classes:

1. General purpose utilities and
2. Matrix-vector utilities.

The general purpose utilities strictly operate on data in the high speed memory HSM (core) and perform tasks like bit manipulation, copying a string of words, simple vector dot product, print a small matrix, etc. The matrix-vector utilities operate on data maintained by the storage manager and frequently use various general purpose utilities in a supportive fashion. A list of the utilities used by SLICE 75 along with brief comments on their purposes is provided in Appendix B.

The overall structure of the SLICE 75 system is illustrated in Fig. 4.1. The remainder of this section is devoted to discussion of the block SLICE 75 shown in Fig. 4.1. It is assumed that the reader is familiar with the operational aspects presented in Section 2 and the mathematical techniques used, as discussed in Section 3. The discussion in this section centers on implementation details.

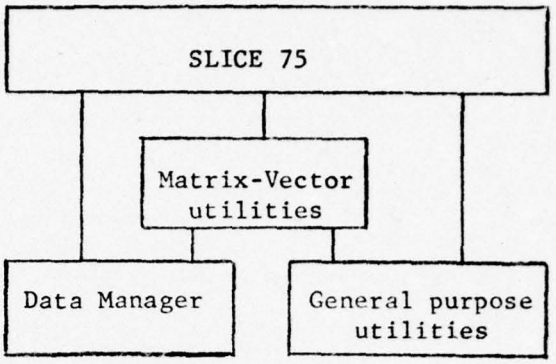


Fig. 4.1 SLICE 75 program system

4.1 System Overview

Processing proceeds in three stages:

1. Set up - set up problem data and working arrays,
2. Subspace- determine a set of vectors which span the desired invariant subspace and
3. Finish - resolve the subspace vectors into eigenvectors and calculate error bounds.

These three stages are fairly independent and, consequently, are written as overlays. Thus, for example, fairly complex free-field input programs and preliminary analysis routines are incorporated into the set-up without adverse effects on the overall program size. Table 4.1 lists the names of the non-utility routines by overlay.

The executive SLIC75, is a very small program that simply causes stages 1, 2 and 3 to be executed in succession. There are good arguments favoring the forming of each stage as a separate program and then executing them separately, in spite of some of the disadvantages which then arise. Although we have chosen to operate with the executive SLIC75, it is clear from its simplicity that the alternative system could be constructed quite easily. The primary advantage of the present system is that the loaded form of the program (absolute) is substantially smaller. The alternative system requires duplication of most of the routines in overlay 0, which includes many support routines not listed in Table 4.1.

Table 4.1 SLICE 75 routines by overlay. Overlay 0 is the master (executive) which is always resident.

No.	Name	Purpose
		Overlay 0
1	SLIC75	Executive control program
2	CNTRL	Sense setting of a control switch
3	INITSM	Initialize the storage manager
4	LODSIM	Load vectors into iteration set
5	ORTHMT	Form an orthonormalization matrix
6	ORTHOG	Orthonormalize the vector iterates
		Overlay 1
7	SETUP	Problem set up executive
8	CENTRLF	Flip and/or display control switches
9	INFORM	Display problem definition data
10	PARMCH	Make specified parameter changes
11	PROBLM	Establish problem definition
		Overlay 2
12	SUBSPC	Subspace determination executive
13	ANALYZ	Analyze completeness of subspace vector set
14	ANALYZ1	Analyze eigenvalue-shift point relationship
15	ASTEP	Take one accelerated iteration step
16	CONVRG	Analyze iteration convergence
17	ITERAT	Simultaneous iteration executive
18	SETBND	Set eigenvalue bound estimates
19	SHIFT	Shift the eigenvalue spectrum
20	SPORTH	Special, new vector orthogonalization
		Overlay 3
21	FINISH	Solution completion executive
22	CLEAR	Clear scratch records from data base
23	ERRBND	Calculate eigenvalue error bounds
24	RESOLV	Form eigenvectors from subspace vectors

4.1.1 Global Records

There is a collection of global records which are used throughout the SLICE 75 system. Although each is not always used for one unique purpose, the usage is generally closely related to the initial purpose. These records are listed in Table 4.2. There are, of course, many temporary records which are created and deleted as required for scratch space and other needs.

The only thing that really distinguishes a global record from any other record is the fact that its names (external/internal) are in the table of contents, record CNTNTS. Thus, if one wants record "M", he must first look up its corresponding internal name, which is constructed by the storage manager at the time the record is declared, and present that to the storage manager. External names are not required for temporary records.

A fundamental requirement of the storage manager is that each record declared must fit in the HSM (high speed memory or core) storage pool. However, in the operation of SLICE 75 we expect to operate on many arrays that do not fit in HSM. For example, the (sparse) stiffness matrix S generally will not fit in HSM. Consequently, records like S, M, SIMVEC, SUBVEC, etc., can simply be "header" records which name other records, each of which contains part of the actual numerical data. For example, if S is extremely large, it can be maintained in a nested fashion as header records, each of which names a set of records containing parts of the numerical data. A similar form of record heirarchy for handling a large matrix is occasionally called a hypermatrix.

Since SLICE 75 never directly deals with the contents of records which are potentially header records, it does not need to take this consideration into account. It simply presents the named record to a matrix-vector utility which is then responsible for properly processing it. The conventions used for general matrices and vectors are given in Appendix B. Certain parameters, however, which are required for the matrix processors are established in the global parameter list discussed below.

Table 4.2 Global records by external name
with location in record CNTNTS

Name	Loc.	Purpose
AGE	11	Ages of simultaneous iteration vectors
BOUNDS	20	Eigenvalue bound estimates and signs
CHEB	9	Table of Chebyshev coefficients
CNTNTS	1	Table of external and internal record names
EIGVEC	21	Matrix of calculated eigenvectors
EIGVL	10	Table of eigenvalues
FSM	5	Factored form of matrix S-MU*M
ISIMVC	16	Set of initial iteration vectors
M	4	Matrix M, commonly a mass matrix
MISIMV	17	Set of initial vectors multiplied by M
MSIMVC	15	Set of vector iterates multiplied by M
MSUBVC	13	Set of subspace vectors multiplied by M
ORTHT	19	Orthonormalization matrix for SIMVEC
PARAM	2	Table of operational parameters
S	3	Matrix S, commonly a stiffness matrix
SHIFTL	8	Table of actual spectral shift points
SHPTS	6	Table of prescribed shift points
SIMVEC	14	Set of simultaneous iteration vectors
SUBVEC	12	Set of subspace vectors
TEGSH	7	No. eigenvalues less than each shift point
YTMY	18	Congruence transformation (SIMVEC, MSIMVC)

4.1.2 Global Parameters

A table of all the parameters for controlling the detailed operations in SLICE 75 is maintained in record PARAM. A default value, for each parameter requiring one, is set by block data when SLICE 75 is loaded. Provision is made, see Section 2, for changing any of the parameter values via input data; however, it is seldom required and must be done with extreme caution if a successful analysis is desired. The parameters are listed in Table 4.3.

In the remainder of this subsection we call attention to rolls played by certain key parameters listed in Table 4.3. The rest are discussed in connection with specific subprocessors.

The fixed parameters are parameters set up by SLICE 75 for processing and are not to be changed by the user from run to run. The standard header sizes used by the matrix utilities for matrices (MHEAD) and vectors (VHEAD) are required for coordination between SLICE 75 and the utilities.

The rest of the parameters are used in assorted ways, i.e., some are constants, some are variables set initially and some are changed continually during the processing. Real parameters ALPHA and BETA, of course, are supplied by the user to define the desired section of the spectrum. From this information, a separate table of shift points is constructed and, at any point in the processing, the shift point in current use is MU.

The size of the problem being solved is given by NUNK (number of unknowns). The group and table of contents ID's of the file from which the matrices S and M are obtained by SLICE 75 are given by IGROUP and ICNTNT. All of the control flags are held right justified in FLAGS. BPFLAG is the bit position (left to right) of flag 0, which is not used. The function of each flag is specified in Table 2.1.

4.1.3 Error Trace System

The subroutines in the SLICE 75 system that use the storage manager call a subroutine PUSHER when first entered and POPER just before the exit. At

Table 4.3 Global parameters
(record PARAM)

Name	Loc.	Purpose
		---Fixed Parameters (Integer)---
LCIP	6	Location of integer parameters in storage pool
LCRP	7	Location of real parameters in storage pool
LPARM	4	Location of record PARAM in storage pool - 2
MAXDIM	3	$= 2^k$ where $2^k \geq$ (max. problem size) $> 2^{k-1}$
MXPARM	5	Capacity of record PARAM for parameters
MHEAD	1	Length of all matrix headers
VHEAD	2	Length of all vector headers
		---Real Parameters---
ALPHA	11	Lower bound of desired spectral section
BETA	12	Upper bound of desired spectral section
BMU	16	Lowest shift point used so far
EPSMCH	14	FLOAT (1 + EPSMCH/2) = 1
GAMMA	10	Subsection definition factor
MU	8	Current spectral shift point
ORTHFC	13	Range factor for vector orthogonalization
PI	18	3.1415926535898
PMU	19	Previous spectral shift point MU
SIGMA	9	= MIN (BETA-MU, MU-ALPHA)
TMU	17	Highest shift point used so far
TOLMCH	15	Iterate convergence criterion (see CONVRG)
		---Integer Parameters---
BPFLAG	26	Bit position ahead of the switch field (FLAGS)
DONE	35	Subspace complete indicator (= 1 if complete)
FLAGS	25	Program control switches
ICNTNT	44	Input group contents table identification
IGROUP	43	Input record group identification
ISIMP	36	Pointer to last initial vector iterate used
ITPORT	45	Min. no. iterations per orthonormalization
MAXEV	30	Max. no. of eigenvectors expected
MAXITR	33	Max. no. of iterations for convergence (ITERAT)
MAXSHP	32	Max. no shift points expected
MAXSIM	31	Max. no initial vector iterates available
NCHEB	28	No. Chebyshev coefficients for iterate acceleration
NCONV	34	No. converged vectors in SIMVEC
NITER	46	Current max. age of a vector in SIMVEC
NOMU	38	Current no. eigenvalues greater than MU found
NOTMU	40	Current no. eigenvalues greater than TMU found
NSHIFT	41	Current no. of spectral shifts taken
NSIMIT	29	No. vectors used in simultaneous iteration
NSIMTR	42	Reference value of NSIMIT
NSVEC	20	Current no. subspace vectors found
NUBMU	39	Current no. eigenvalues less than BMU found
NUMU	37	Current no. eigenvalues less than MU found
NUNK	27	Problem size (No. of unknowns)
SIDE	24	Position indicator for shift point table SHPTS
TEGBMU	22	No. eigenvalues less than BMU
TEGMU	21	No. eigenvalues less than MU
TEGTMU	23	No. eigenvalues less than TMU

various points in such subroutines, an integer variable, usually called SOURCE, at the beginning of labeled common /ERRCOM/ is set to an alpha-numeric name indicative of which part of the subroutine is active.

The purpose of these antics is to provide a useful trace back system in case of failure in one of the utilities. An array in /ERRCOM/ operates as a stack with SOURCE at the top. PUSHER pushes the stack down so that new information can be placed in SOURCE and, of course, POPER pops the stack up.

4.2 Processing Details

In this section we discuss the three stages of processing used by SLICE 75. Sufficient detail is provided to give the user a qualitative understanding of what key tasks are being accomplished in each of the three stages mentioned in Section 4.1 and by which subprocessors they are accomplished. This discussion, together with the listings, should provide the interested investigator with all the information he needs in a reasonably painless fashion.

4.2.1 Set-Up

The tasks accomplished during this stage are listed in Table 4.4 in the order in which they are carried out. The program flow is a simple sequence under the control of SETUP.

The setting up of the problem information is controlled by PROBLM as indicated. It utilizes subroutines CNTRL, CNTRLD, CNTRLF and PARMCH as well as the general data input routines for setting parameters from user input. It uses the record and group managers for finding matrices S and M. Finally, it uses INFORM to display all requested initial data.

4.2.2 Subspace Determination

This stage is, of course, by far the most complex part of the entire process. It is the implementation of the algorithm outlined in Section 3.1.1. In this section we shall frequently refer to steps in that algorithm. The overall process is controlled by subroutine SUBSPC. This fairly simple routine divides

the outer loop on spectral shift points into three parts: spectral shift, iterate and analyze. These substages are under the control of routines SHIFT, ITERAT and ANALYZ.

4.2.2.1 Spectral Shift

The selection of a new shift point is not carried out in this part. The shift point is initialized in SETUP and selected as required in part 3, analyze.

The main task performed by SHIFT is, given a new shift point μ, form and factor the matrix

$$S' = (S - \mu M) .$$

If it happens that S' is singular, increase μ by an amount $.0001*\sigma$ and try again, where σ is the range factor. Provision is made for up to 3 changes in μ before the program gives up.

A second task performed by SHIFT is a scaling of the Chebyshev polynomial coefficients c_j, see Section 3.1.1, so that the acceleration coefficients $c_j \sigma^j$ used in step 3 of the algorithm do not get too large (or small).

Finally, SHIFT performs the bookkeeping task of recording the current shift point and setting related parameters. An important parameter for SHIFT is the previous shift point PMU. Occasionally SHIFT is called with MU = PMU in which case, of course, SHIFT does nothing.

4.2.2.2 Iteration

This part of the subspace determination loop includes steps 2,3,4,5 and 6 of the algorithm of Section 3.1.1. The overall process is under the control of routine ITERAT which delegates subtasks to subprocessors as discussed below. The discussion is organized by steps in the algorithm of Section 3.1.1. The subroutine lists do not include CNTRL, POPER, PUSHER and storage manager routines.

Steps 2 and 3. Accelerated Iteration Step.

Subroutines: ASTEP, MMPY, MSCL, MSOL, MSUM, ORTHOG, SCOPY, SETBND

The details for these steps are contained in ASTEP, which utilizes all of the subroutines listed in the process. Actually, many more routines are involved which are called by those listed. Of course, the sum required by step 3 is incorporated into the iteration phase of step 2 to avoid saving the intermediate iterates $Z^{(j)}$.

In addition to the explicit tasks outlined in steps 2 and 3, ASTEP calculates the bounds required for the vector acceptance tests as discussed in Section 3.4. These results are placed in an NSIMIT by 5 matrix called BOUNDS. The first column of BOUNDS, which normally contains estimates of the eigenvalues corresponding to the simultaneous iterates SIMVEC, is initially copies into column 4 for retention. Then, assuming the number of iterations in step 2 of the algorithm is k (NCHEB in the program), the bounds corresponding to iterate $Z^{(k)}$ in column 2 and $Z^{(k-1)}$ in column 1 of BOUNDS. The numerators of the Rayleigh quotients for $Z^{(k)}$ are placed in column 5 of BOUNDS as indicators of the signs of the corresponding eigenvalues. The three sets of bounds are required for the bound estimate acceleration process used for the vector acceptance test by routine CONVRG, see Section 3. The bound estimates derived from column 3 are later placed in column 1 of BOUNDS by routine CONVRG.

SLICE 75 does, however, permit the use of Chebyshev acceleration of order less than 3, which corresponds to $k = NCHEB < 4$. This fact complicates ASTEP and CONVRG somewhat, since alternative convergence criteria must be used. When $k = NCHEB < 4$, the bound corresponding to the accelerated iterate is placed in column 1 of BOUNDS by ASTEP.

Steps 4 and 5. Orthogonalization

Subroutines: ORTHOG, EIGSYM, MMPY, MPRINT, MSCL, MTMPY, MTMPYS, ORTHMT, SCOPY, VSUM

The general orthogonalization routine ORTHOG is a fairly complex program that performs combinations of the following tasks:

A. Orthogonalize the simultaneous iterates SIMVEC with respect to the subspace vectors SUBVEC, using MSUBVC for achieving M orthogonality,

B. Multiply SIMVEC from Task A by matrix M to produce MSIMVC,

C. Orthonormalize (M orthogonality) the iterates in SIMVEC using MSIMVC.

All of the listed subroutines are directly called upon by ORTHOG for support. The routine ORTHMT produces the matrix QD^{-1} shown in step 4 of the algorithm of Section 3.1.1.

Control over which tasks are carried out is through the first argument CASE (integer) of routine ORTHOG. Table 4.5 shows the interpretation of CASE.

Table 4.5 Interpretation of CASE Parameter for Orthogonalization

CASE	TASKS
less than -1	A, B
-1	A
0	A, B, C
greater than 0	C

Step 6. Acceptance Tests

Subroutine: CONVRG

CONVRG simply applies the convergence tests for each of the iterates SIMVEC by examination of the data in matrix BOUNDS, see discussion under steps 2 and 3. It notes how many passed (parameter 34, NCONV) and places the error estimates in column 4 of BOUNDS. Conservative estimates of the eigenvalues, based on the bounds for the last iterates ($Z^{(k)}$ of step 2) appearing in column 3 of BOUNDS, are placed in column 1 of BOUNDS for general applications.

As long as the number of iterations NCHEB per accelerated step (see discussion under Steps 2 and 3) is greater than 3, the standard vector acceptance step discussed in Section 3.4 is used. If NCHEB $\leq$ 3, however, then the current bounds (column 1 of BOUNDS) are used to calculate eigenvalue estimates which are then simply compared with previous estimates (placed in column 4 of BOUNDS by routine ASTEP) and convergence is assumed when the change is negligible.

Notice that CONVRG does not carry out all the operations specified under step 6. The last three items, which are strategy decisions, are carried out by ANALYZ which is discussed below.

4.2.2.3 Analyze Convergence

Subroutines: ANALYZ, ANALZ1, LODSIM, SPORTH

The most nebulous and difficult part of the subspace determination process is analyzing what has been achieved at each point in the process and making the appropriate decisions for continuation. This task involves parts of step 6 and all of step 7 in the algorithm presented in Section 3.1.1. This entire task is carried out under the control of ANALYZ which calls upon the listed subroutines for support.

The steps taken by ANALYZ are in order:

1. Append the converged iterates (if any) in SIMVEC to the set of subspace vectors SUBVEC. Simultaneously do the same for MSIMVC and MSUBVC, and append the corresponding eigenvalue estimates (in column 1 of BOUNDS) to the table EIGVL of approximate eigenvalues.
2. Delete redundant vector iterates (vectors determined to be not linearly independent of the rest by ORTHOG) from the sets SIMVEC and MSIMVC.
3. Replace the vector iterates removed from SIMVEC and MSIMVC in 1. and 2. above by new initial iterates from ISIMVC and MISIMV, and set the ages of these new iterates to zero.
4. Determine if the subsection $(\mu-\gamma\sigma, \mu+\gamma\sigma)$ is exhausted.

5. If the subsection is exhausted, determine if the whole section is complete. Include in this test a comparison of the theoretical number of eigenvalues between extreme shift points (BMU and TMU) with the actual number found. If a conflict is found, determine (in routine ANALYZ1) between which adjacent shift points an eigenvalue is missing and set the new shift point to the middle of that interval. If no conflict, set parameter DONE = 1.

6. If the subsection is not exhausted, determine a new shift point.

The above outline of steps taken by ANALYZ is also provided in the source code listing. Further details on these steps is most conveniently obtained from the source code.

As soon as the parameter DONE is set to 1 by ANALYZ, the subspace corresponding to the requested interval (ALPHA, BETA) of the eigenvalue spectrum is taken to be complete. The dimension of the subspace is given by parameter NSVEC and the vectors are in set SUBVEC. These vectors are M orthnormal and the vectors M*SUBVEC are stored as set MSUBVC.

4.2.3 Finish

The last of the stages introduced in Section 4.1 is by far the simplest of the three. It is controlled by the three FORTRAN statement routine FINISH which calls routines RESOLV, ERRBND and CLEAR in that order.

RESOLV resolves the subspace vectors SUBVEC into eigenvectors in the straightforward manner indicated in step 8 of the algorithm of Section 3.1.1. It then sorts them and the eigenvalues (in order of increasing eigenvalues) and displays the final results. It requires subroutines EIGSYM, MMPY, MPRINT, MTMPY, SCOPY, SORT and THYME as well as CNTRL, storage manager routines and the typical utilities PUSHER, etc.

If error bounds are requested (flag 7, see Table 2.1), ERRBND is called which produces rigorous eigenvalue error bounds by the simple process discussed briefly in Section 3.4. These error bounds are stored at the beginning of record BOUNDS. ERRBND uses subroutines MMPY, MSOL, MSUM, MTMPYD, SCOPY and THYME along with the typical routines CNTRL, etc.

Finally, routine CLEAR eliminates all the unnecessary records from the record table and catalogues a new group consisting of the important results of the SLICE 75 eigenvalue analysis. Table 4.6 lists the records that are retained in the final record group.

Table 4.6 Final Catlogued Records

Name	Contents
CNTNTS	Table of external/internal names of catalogued records.
PARAM	Table of final parameter values.
S	Access record for matrix S.
M	Access record for matrix M.
FSM	Access record for the factored matrix $(S-\mu M)$ at the final shift point μ.
EIGVL	Table of final eigenvalues.
BOUNDS	Table of eigenvalue error bounds.
EIGVEC	Access record for the final eigenvectors and the vectors themselves.

Note in Table 4.6 that some of the records are called access records. That means that they are records which provide access information to the data but may not contain the data. For example, EIGVEC specifically names a series of records, each of which contains one actual vector. S, M and FSM may or may not contain the actual matrix data, depending upon the particular matrix types involved.

Appendix A
STORAGE MANAGER FUNDAMENTALS

The AID (Analysis Information Description) storage manager is a simple, flexible system for managing both temporary and permanent storage of data in FORTRAN application programs. It is oriented toward the data structures used in engineering and scientific analysis programs as opposed to accounting and text retrieval programs.

The five key processors in the AID system are illustrated in Figure A.1. The HSM (high speed memory or core) manager MGRHSM is a simple, independent routine for managing a couple hundred records in a storage pool in HSM. If this type of management is all that is required, an application can effectively utilize MGRHSM without any of the other programs.

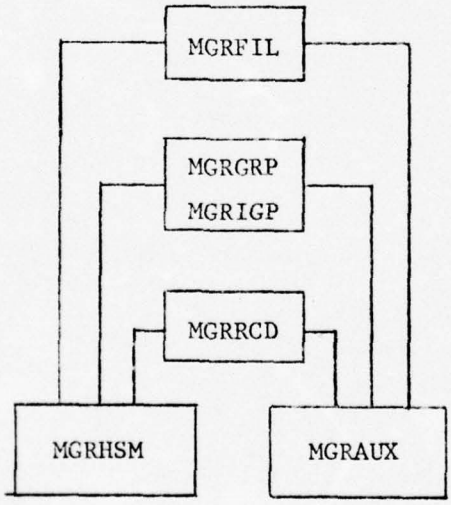


Fig. A.1 Key Storage Management Processors

The auxiliary storage manager MGRAUX is a simple, independent routine for opening, closing, positioning, storing on and copying from mass storage devices. MGRAUX can also be used independently of the rest of the AID system.

Finally, MGRFIL is provided for the final cataloguing operations for a process involving storage mangement. Its main function is to store the GAT (group access table) and identify it (printed as the File ID). It also does file closing operations to insure that the mass data is made a permanent file on the system.

A.1 HSM Management

All high speed memory management statements are in the form of subroutine calls. The subroutine name, which is actually an entry point in the subroutine MGRHSM, is always of the form xxxHSM, where the sequence xxx indicates the function to be carried out. The subroutine calls always have three integer arguments (NAME, LOC, SIZE), even though not all are needed for certain management functions. The management statements are summarized in Table A.1.

Table A.1 High Speed Memory Management Statements

Statement Form: CALL xxxHSM(NAME, LOC, SIZE)
The argument source in the table is indicated by U - user, M - manager, and D - dummy argument.

xxx	Purpose	NAME	LOC	SIZE
AVL(1)	Provide Available Storage in Pool	D	M	M
CHS	Change a Record Size	U	M	U
CMP(1)	Compress Storage	D	M	M
DCL	Declare a Record	M	M	U
DLT	Delete a Record	U	D	D
FND	Find a Record	U	M	M
MGR(2)	Initialize Manager	D	U	U

(1) AVL and CMP set LOC to the largest location currently being used and SIZE to the current available space.

(2) MGR requires the blank common starting location LOC for the pool and the space SIZE in labeled common for record access tables, see Sec. A.1.1.

A.1.1 Management Technique

Consistent with the design of most general purpose digital computers, the storage pool may be viewed as one large vector. In FORTRAN programs it is convenient to use blank common for this purpose. Thus in the user main program, blank common might typically be declared as follows:

```
REAL            USER(500)
COMMON          NCOM, COM(5500)
EQUIVALENCE     (USER, COM)
NCOM = 5500
```

where COM(501),COM(502),...,COM(5500) is to be the storage pool and USER, reserved for some particular user application, is not to be a part of the pool. NCOM is to be set one less than the total defined common area as indicated. The exclusion of USER from the storage pool in the above example is accomplished by initializing the manager with a starting location of 501.

Record identification consisting of a unique name, a blank common location, and a size (NAME, LOC, SIZE) is associated with each declared record. This information is entered in a HSM access table (HAT), which is thus dimensioned HAT(3,HATSIZ), where HATSIZ is the maximum number of records the user expects to ever have declared at one time. HAT could be a part of the storage pool; however, to protect it from accidental user overrun, it is declared by the user main program in a special labeled common block/HSMCOM/.

The name assigned to a record by the manager is unique and permanent, whereas the location and size may vary. Thus manager access to a record is always by name. To facilitate this access, the index of the record identification in the HAT is imbedded in the name, thus avoiding the need to extensively search the HAT for a given name.

A.1.2 Communication Technique

For convenient communication with the user program, several parameters are included in the labeled common /HSMCOM/ in addition to INUSE,NALC and the HAT. Specifically, it is declared as

```
      COMMON /HSMCOM/ HCOND,NCMP,USED,USING,INUSE,NALC,HATSIZ,HPSN,
     *                NIL,HIGH,HAT
```

where HCOND is a condition flag set after each management operation, NCMP is the (negative) number of storage compressions made since last being reset, USED is the maximum pool storage used to date, USING is the pool storage currently in use, HPSN is the HSM location of the most recently accessed record, and HIGH is the largest HSM location used to date. HATSIZ is the record capacity of HAT and NIL is the name assigned to a deleted record. The other parameters appear in Fig. A.2.

A.1.3 Termination Conditions

There are six conditions in which the manager can terminate, four of which are failure conditions. These are indicated to the user by the value in the first word (HCOND) in labeled common. These conditions are summarized in Table A.2.

The manager maintains a pointer called INUSE which indicates the last location in the storage pool which is assigned to some declared record as illustrated in Fig. A.2. A similar pointer NALC is maintained for the HAT. When a new

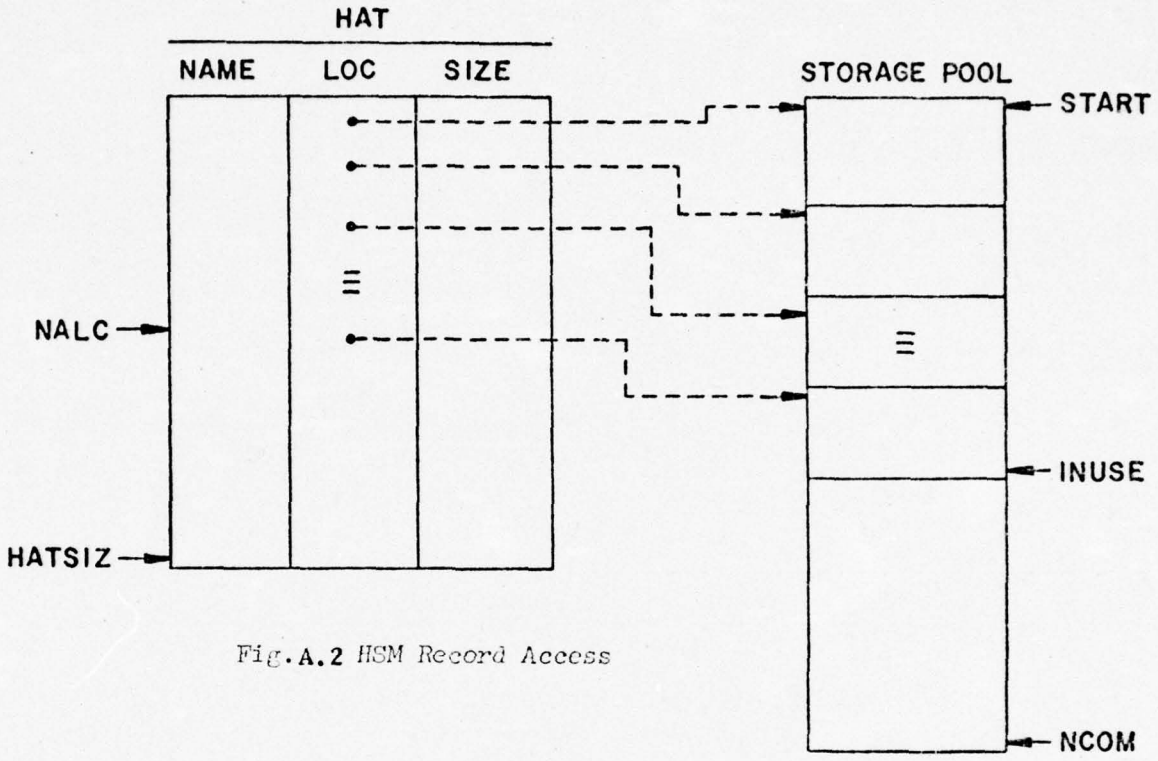


Fig. A.2 HSM Record Access

record is declared, the SIZE is compared with (NCOM-INUSE). If SIZE is smaller or equal, then LOC is simply set to INUSE+1 and INUSE is replaced by INUSE+SIZE. If SIZE is larger, then a flag is checked which tells if any records have been deleted or reduced in size. If none have, then the manager indicates storage overflow and returns.

When a record is deleted, its name entry in the HAT is set to -NIL-, and no further action is taken unless it is the last record. When the last declared record is deleted or reduced in size, the pointers INUSE and NALC are immediately adjusted accordingly. When deleted space in the pool is needed, such as when the SIZE of a declared record exceeds (NCOM-INUSE), then a storage compression is initiated which moves the active (non-deleted) records up in the storage pool, covering the deleted records. It similarly moves up the active entries in the HAT. The pointer INUSE and NALC are then appropriately moved giving a new, larger value of (NCOM-INUSE) for comparison with SIZE. Storage compression may also be invoked by a compress statement.

Table A.2 HSM Manager Stop Conditions

The integer HCOND is the first word of labeled common /HSMCOM/. The HAT is the high speed memory access table

HCOND	Meaning	Pertinent Statements
-1	Storage pool was compressed	DCL,CHS,CMP
0	Statement properly executed	All
1	Record name not in HAT	CHS,DLT,FND
2	Record name already in HAT	DCL
3	HAT overflow	DCL
4	Storage pool overflow	CHS,DCL
5	Record size smaller than 1	CHS,DCL

A.2 Auxiliary Storage Management

The three basic operations required to use an auxiliary storage device are position, store, and copy (or read). Since the details of specifying these operations vary among computer systems, it is convenient to utilize a machine dependent program MGRAUX to translate the specifications used by the machine independent storage manager to those of the specific computer system used. To do this, MGRAUX maintains the following information in array DEV for each auxiliary storage device used:

1. NAME - Device name or number for access
2. LOC - Current location
3. NEXT - Next free location
4. LIMIT - Final available location on the device

This information is kept in labeled common storage

```
COMMON /AUXCOM / ACOND, NDEV, RDEV, DEV(4,NDEV)
```

where NDEV is the total number of devices of which the first RDEV are randomly accessible. ACOND is a condition flag set by MGRAUX after each auxiliary storage operation. Any other (machine dependent) information, e.g., sector sizes, must be maintained internally by MGRAUX.

A.2.1 Auxiliary Storage Operations

The program MGRAUX can conveniently be written as one subroutine with several entry points. The following entry points are required by the higher level managers which are described in Sections 4 and 5:

```
PSNAUX (DUMMY, DEVICE, LOC)
STOAUX (ARRAY, DEVICE, SIZE)
COPAUX (ARRAY, DEVICE, SIZE)
```

The argument DEVICE is the number of the column in the table DEV (i.e., DEV(I,DEVICE), I = 1,4) in which descriptive data for the device is kept.

The operation STOAUX (COPAUX) simply moves data from (to) array ARRAY in high speed memory to (from) the current location of the specified device. The current location, i.e., DEV(2,DEVICE), is left pointing at the beginning of the next record or free space on the device.

If the sign of DEVICE is positive, the operation PSNAUX positions the specified device to position LOC (in computer words) if LOC is a valid position. The device is positioned to the beginning if LOC is too small or to the next available (free) location if LOC is too large. If addressing is by sectors and LOC leads to a fractional number of sectors, e.g., 2.3, then DEVICE is positioned to the beginning of the next sector, e.g. 3.

If the sign of DEVICE is negative, PSNAUX moves the current location ahead by one record of size |LOC|, in computer words, when LOC is positive and backward one record when LOC is negative. Again, for sector addressed devices, positioning is over a sufficient number of sectors to cover the specified record size.

If DEVICE = 0, then the device number and location are determined from the LOC

as follows, using integer arithmetic:

LOCATION = LOC/64

DEVICE = LOC-64*LOCATION

A.2.2 Failure Conditions

Depending upon the computer system, there can be a variety of causes for failure of an auxiliary storage operation. Of these, the following three conditions have been found useful for the higher level management processors:

Condition	Meaning
1	Undefined auxiliary storage device,
2	End of information encountered by copy,
3	Device overflow.

A.3 Record Management

Like high-speed memory management, the objective of record management is to provide portions of the HSM storage pool to an application program as required. However, it pertains to quantities of data which, generally, far exceed the HSM capacity. Consequently, the manager MGRRCD associates a priority value with each record and holds low priority records on auxiliary storage to make room in HSM.

The record manager consists of several support routines for a primary subroutine MGRRCD which has the management statements as entry points. Included among the support routines are the high-speed memory manager MGRHSM and the auxiliary storage manager MGRAUX as illustrated in Fig. **A.1.**

A.3.1 Management Statements

Because auxiliary storage devices are involved in general record management, more management statements are required than for HSM management. A list of the management statements used is provided in Table **A.3.**

Table **A.3** Record Management Statements

Statement Form: CALL xxxRCD(NAME, LOC, SIZE)
The argument source in the table is indicated by U - user, M - manager, UM - optional user input which may be changed by the manager and D - dummy argument

xxx	Purpose	NAME	LOC	SIZE
CHS	Load in HSM, delete aux. copies and change size	U	UM(1)	U
DCL	Make space in HSM for new record	M	M	U
DLH	Delete HSM copy of record	U	D	D
DLT	Delete record	U	D	D
FND	Load record in HSM	U	UM(1)	M
LOC	Locate a record in HSM or aux. storage	U	M(2)	M
MGR	Initialize record manager	D	D	U(3)
PRG	Purge deleted records from all storage	D	D	U(4)
PRI	Set record priority	U	U	D
SAL	Save all records in HSM in aux. storage	D	UM(5)	U(6)
SAV	Save named record in aux storage	U	UM(5)	U(6)

(1) A record may be loaded into HSM on top of a suitably sized record, RH say, currently in HSM by operations CHS and FND, by giving the name RH as LOC.

(2) If the record is not in HSM, LOC is set to the negative file position, -(64*Location + Device).

(3) SIZE is the space in labeled common for record access data, see Sec. 4.2.

(4) If SIZE = -1, purge random auxiliary storage as well as the RAT.

(5) LOC indicates on which auxiliary storage unit data is to be saved for operations SAL and SAV. The actual storage location (64*Location + Device) is returned.

(6) If SIZE = -1, the HSM copy of the record is deleted after the record save operation.

A.3.2 Communication

For each declared record, the following information is entered in a record access table (RAT):

NAME -	The unique record name,
SIZE, PRIORITY -	The size of the record and its priority (64*SIZE + PRIORITY)
HNAME -	Its HSM name if it is in HSM, otherwise an internal value for -NIL-,
RAUX -	Its location on random access auxiliary storage (64*Location + Device No.)
SAUX -	Its location on sequential access aux. storage

Thus the RAT is dimensioned RAT(5,NRCD), where NRCD is the maximum number of records that can be simultaneously declared.

Similar to the HSM manager, the access table RAT along with several communication parameters is kept in a separate, labeled common /RCDCOM/ for protection. Specifically it is declared (except for the dimensions which must involve integers) as follows:

```
      COMMON/RCDCOM/ RCOND, NFAIL,PRIRTY,NIL,MRAT,BUMPED,BUMP(5),
     *               GROUP,NRCDS,LARGE, RAGSIZ,RATSTP,MRDEV,AUXLOC(MRDEV),RAT(5MRAT)
```

All of the parameters are integers and, to some extent, are simply used to facilitate communication between MGRRCD and its support programs. The parameters which are useful to the user (application program) are as follows:

RCOND -	The manager condition (set after each management operation),
NFAIL -	The sum of the positive failure conditions generated (may be reset by the user),
PRIRTY -	The priority of declared or most recently accessed record,
MRAT -	The record capacity of the RAT,
MRDEV -	The dimension of array AUXLOC
BUMPED,BUMP -	The number (less than 5) and names of the records most recently removed from HSM in order to create more HSM storage space.

AUXLOC -	The initial positions of the RDEV random access auxiliary storage devices (see MGRAUX).
GROUP -	The name of the record group currently being treated,
NRCDS -	The no. of record positions in the RAT currently set,
LARGE -	The size of the largest record declared to date.

A.3.3 Record Priority

When declaring a record, its priority is set to the current value of PRIRTY. This is particularly convenient when several records of the same priority are to be declared. Whenever a record is found using LOC, FND, or CHS, its priority value is given to PRIRTY. When setting a record priority by the PRI statement, the second argument (LOC) is used as the priority value.

A.3.4 Record Access

Similar to the HSM manager, the record manager accesses each record by its unique name. As in the MGRHSM, this access is facilitated by imbedding the initial RAT index of the record in the name. Also, when a record is deleted, its name is simply set to the value NIL and the HSM copy, if it exists, is deleted. However, if it is the last (most recently declared) record, NRCDS is reduced by 1 in addition, thus purging the record from the RAT.

A.3.5 Bumping Records

When more HSM storage space than is currently available is required for an operation such as DCL, FND, or CHS, the records currently in HSM are sorted by priority. Then, starting with the lowest priority, the records are removed from HSM (bumped) and placed in auxiliary storage until sufficient HSM space is released. If N records are thus bumped, then the parameter BUMPED is set to N mod 5 (or 5 if N is a multiple of 5) and the names of the last BUMPED records are left in the parameter array BUMP in labeled common storage.

A.3.6 Auxiliary Storage of Records

Bumped records are stored sequentially on the first specified auxiliary storage device until it is full, then on the next device and so forth. This stor-

age pattern also applies to the operations SAL and SAV in the absence of a unit specification in the second argument, see Table A.3.

If the size of a record is increased, then auxiliary storage address is deleted. If it is subsequently saved, it is placed in the next available position in auxiliary storage instead of its previous position. Thus the statements CHS and DLT can cause the existence of "gaps" in auxiliary storage consisting of obsolete data. On random access devices, these gaps can be eliminated by the purge process discussed in the next subsection.

Records that are of a permanent nature, e.g., are to be used by other programs, can be stored on sequential devices by specifying the proper device number in the SAL or SAV statement. On these devices, gaps are never eliminated by the manager. The only other way records will be placed on sequential devices is in the event of overflow of all the random devices. In this case, operations SAV, CHS, etc. will use the sequential devices without argument specification. This is a necessary hazard since on some computer systems, random devices are not available and such "spill over" is necessary and desirable.

A.3.7 Purging

All deleted records are purged from the RAT by the PRG statement or, if the RAT becomes filled and DCL statement is subsequently issued.

A.3.8 Stop Conditions

The record manager can prematurely terminate due to problems in the lower level processors MGRHSM and MGRAUX as well as problems in MGRRCD. It has been found convenient for problem tracing to assign record manager condition numbers to some of the former causes as well as the later causes, even though it results in some redundancy. The record manager stop conditions are summarized in Table A.4.

Table A.4 Record Manager Stop Conditions

RCOND is the first word of labeled common /RCDCOM/. The RAT is the record access table.

RCOND	Meaning	Pertinent Statements
-1	Deleted records were purged	DCL,PRG
0	Statement properly executed	All
1	Record name not in RAT	CHS,DLH,DLT,FND,LOC,PRI,SAV
2	Record name already in RAT	DCL
3	RAT overflow	DCL
4	HSM storage pool overflow	DCL,PRG
5	Record size smaller than 1	CHS,DCL
6	Record is not in HSM	DLH,SAV
7	Auxiliary storage overflow	CHS,DCL,FND,SAL,SAV

There are some improper statements, such as setting a priority out of range or attempting to save a record on an undefined device, which could be rejected by the manager and assigned an error condition number. However, we have chosen to make the manager very forgiving and so, in several cases, when an invalid parameter value is given, the manager assumes the nearest valid value to the given parameter value.

A.4 Higher Level Management

For applications involving more than a couple thousand records, higher level management processors are required as illustrated in Fig. A.1. In a particular application, the number N_R of records that can be handled by the record manager MGRRCD is established by the storage space allocated to labeled common

/RCDCOM/ by the user for the record access table RAT, see Sec. A.3. The term group is used to distinguish a collection of N_R-1 or fewer records defined sequentially in the RAT, starting at the beginning.

The RAT together with the parameters GROUP,NRCDS,LARGE, and AUXLOC for a collection of records is itself a record which is called the RAG (Record for Access to the Group). For each group there is an associated RAG which may either reside in labeled common /RCDCOM/ or in auxiliary storage. When the RAG for a group is in labeled common, the group is said to be active.

A.4.1 Group Management

The group manager MGRGRP keeps track of record groups. When a group is declared, operation DCLGRP, the following sequence of operations takes place:

1. All HSM copies of currently declared records are saved (SALRCD) and deleted from HSM,
2. A record purge is executed (PRGRCD),
3. The record for access to the group (RAG) is declared and the record access table RAT, with its associated parameters and exclusive of the entry for RAG, is stored in the RAG,
4. The RAG is saved(with the HSM copy deleted) and its record name and auxiliary storage location are entered into a group access table GAT, and finally
5. The RAT is cleared and its associated parameters are re-initialized for forming a new group.

The group access table GAT is held in labeled common. For each group, the group name and the auxiliary storage location (device no. + 64 * device location) of its access record RAG is kept in GAT. The form of the labeled common specification is:

```
      COMMON /GRPCOM/ GCOND,MAXGRP, NEWNM,NGRP,MXRCD,
     *                FILSIZ,INDEX,GAT(2,MAXGRP).
```

where GCOND is the condition flag, NGRP is the number of groups currently declared and FILSIZ is the fixed size of /GRPCOM/. The group manager takes the liberty of reducing the recorded size MRAT of the record access table RAT in order to be able to declare record space for the RAG. The true maxi-

mum record capacity is then held in MXRCD. Other labeled common parameters are used for inter-program communication and masking.

A.4.2 Group Management Statements

Records may be deleted or added, up to a maximum of N_R-1, to any group. When a group is modified, the replace operation REPGRP must be executed to update its access table RAG. The group operations are summarized in Table A.5. To date, delete and purge operations for groups have not been required. Should the need arise, however, their inclusion in MGRGRP would be a very simple programming task.

Table A.5 Group Management Statements

Statement form: CALL xxxGRP(NAME)
The argument source is indicated by U-user, M-manager.

xxx	Purpose	Argument
DCL	Declare a new group	M
FND	Find a declared group	U
LOC	Locate a group RAG in aux. storage	UM(1)
MGR	Initialize group manager	U(2)
REP	Replace currently active group	M

(1) The group name is provided by the user and is replaced by the location (64*Location + Device) by the manager.

(2) The argument for MGR gives the labeled common /GRPCOM/ space declared by the user.

A.4.3 Stop Conditions

Similar to the record manager, the group manager may terminate prematurely either because of a failure in a lower processor (MGRHSM,MGRAUX, or MGRRCD) or because of a failure in it. As with the record manager, it has been found convenient to assign group stop condition values for some of the former causes as well as the latter. The group stop conditions are given in Table A.6

Table A.6 Group Manager Stop Conditions

The group condition number GCOND is the first word of labeled common /GRPCOM/. GAT is the group access table and a RAG is a record for access to a group.

GCOND	Meaning	Pertinent Statements
0	Statement properly executed	All
1	Named group is not in GAT	FND,LOC,REP
2	Group name is already in GAT	DCL
3	GAT overflow	DCL
4	HSM storage pool too small for RAG	DCL
5	No declared records for this group	DCL
6	Attempt to declare an established group	DCL
7	Auxiliary storage error	DCL,FND,REP

A.4.4 Inter-Group Communication

The record manager has access only to the records of an active group, i.e., a group for which the RAG is in labeled common. Access to records in non-active groups is provided by an inter-group record manager MGRIGP. This is a support processor for MGRGRP and uses the same labeled common /GRPCOM/ for communication.

The group management philosophy is that only an active group can be changed in any way. Inter-group activity is strictly for obtaining information, i.e., a form of "read only" data retrieval. Records obtained from non-active groups are given new names and treated as new records in the active group. Any opera-

tions on them do not affect the originals. The inter-group management statements are summarized in Table A.7.

Table A.7 Inter-group Management Statements

Statement Form: CALL xxxIGP (NAME,LOC,SIZE)
The argument source in the table is indicated by U - user, M - manager, and D - dummy argument.

xxx	Purpose	NAME	LOC	SIZE
LOC(1)	Locate the named record	U	UM	UM
FND(2)	Load the named record in HSM and assign a new name	UM	UM	UM
DLT	Delete the non-active RAG copy	D	D	D

(1) The user provides the record name (NAME), the group name (LOC) and the group RAG disposition flag (SIZE). SIZE = -1 implies hold the RAG for subsequent access. The manager returns the size and the location, positive if in HSM and negative if in aux. storage.

(2) The user provides the record name (NAME), group name (LOC) and the disposition flag (SIZE). The manager returns the new name (NAME), HSM location, and size.

The failure stop conditions for MGRIGP are GCOND = 1 if the named group is not in CAT and RCOND = 1 if the named record is not in the RAG for the named group.

A.5 Cataloguing Files

When communicating data between application programs which use the AID storage manager, it is necessary to place the declared records in permanent mass storage along with a simple mechnism for retrieving them. In this respect it is helpful to provide meaningful names, which we shall call external names, for those records to be accessed. The names assigned to records by AID will be called internal names.

A currently used technique for establishing external names for N records say, is to declare a "table of contents" record CNTNTS of size 2N + 1. The first entry in CNTNTS is N, followed by the N external names, followed finally by the N internal names assigned by AID forthe records as illustrated in Figure A.3.

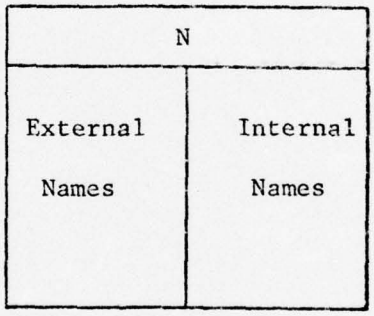


Fig. A.3 Table of Contents Record

Although not required, the table of contents can certainly include its own names, e.g., CNTNTS, as well.

The actual final catloguing is then carried out with three FORTRAN statements:

```
      CALL DCLGRP(NAMGRP)
      WRITE (6,1)  NAMGRP, CNTNTS
      CALL CLSFIL (DUMMY)
1     FORMAT (/9X,22(1H-)/10X,8HGROUP ID,5X,I7
             /10X,11HCONTENTS ID,2X,I7/9X,22(1H-))
```

The call to DCLGRP clears the high speed memory and establishes the group name which is then printed together with the internal table of contents name. The call to CLSFIL then records the group access table and prints an identifier indicating its location.

Appendix B
UTILITY PROGRAMS

Two classes of utility programs have been collected to be used in support of general scientific analysis programs. The most basic are the general purpose utilities which include all utility type routines that operate entirely in HSM (high speed memory). The more complex utilities are the matrix-vector utilities which use the storage manager and general purpose utilities for support.

B.1 General Purpose Utilities

A guiding principle for the design of general purpose utilities is that they should not have any input or output statements other than an occasional error printout if required. This principle has been followed in all of the routines except for the general, free field read routines FFREAD and READR.

A second guiding principle is to avoid the use of common in general purpose utilities. This principle also has been followed in all but the error control routines PTRACE, PUSHER and TYPCHK. These refer to the general error source stack /ERRCOM/.

The general purpose utility routines which directly support SLICE 75 together with its associated matrix-vector utilities are listed in Table B.1.

B.2 Matrix-Vector Utilities

The matrix-vector utilities all operate on standard forms of arrays* which are maintained as records by the general purpose storage management system AID discussed in Appendix A. Every array is represented by one record which contains descriptive information about the array and either the array inself or access information for getting the array.

* In this context, an array is taken to mean either a matrix or a vector.

Table B.1 General Purpose Utilities Supportive of SLICE 75

Note: Names following / are entry points

NAME	ARGUMENTS	PURPOSE
EIGSYM	A,NA,N, EVAL,EVEC, SC	Eigenvalues EVAL and vectors EVEC of real symmetric N by N matrix A. A and EVEC dimensioned NA by N, scratch vector SC dimensioned N.
FACSYM	A,N,EPS, SING, NEGDG	Factor real, symmetric N by N matrix A in place. Integer SING is the number of diagonals in factor smaller than EPS and NEGDG is the number of negative diagonals in the factor.
FFREAD	L, T, N, ST	Read card image into string ST (72A1) and decode into list L of items with types T (0: integer, 1: real, -1: alpha). N is the no. of items found.
FLD	IB, NB, S	Integer function yielding bit field (IB, IB + NB - 1) of S right justified. The left bit of S is numbered 0.
FLDPAC	IB, NB, D, S	Insert the right NB bits of S into field (IB, IB + NB - 1) of D.
MPYAA	P,A,B, L,M.N, SYM	P=A*B where A is L by M and B is M by N. If P will be symmetric, set SYM= .true.
MTMPAA	P,A,B, LMN,N, ICASE	$P=A^T*B$ where A is M by L and B is M by N. If P will be symmetric, set ICASE=1. If only diagonals of P are wanted, set ICASE= -1. Otherwise set ICASE=0.
PRMTX	A,M,N,ITYPE	Print M by N matrix A of type ITYPE (4 - upper triangular, 5 - lower triangular, 6 - full).
PTRACE	NAME, R1,R2,R3	Record activity trace routine. NAME - calling routine; R1,R2,R3 - record names involved. Uses labeled common / ERRCOM /.
PUSHER POPER		Push down or pop up the trace back stack in labeled common / ERRCOM /.
READR	N,R,S,IS	Read N real numbers into R using FFREAD. Scratch arrays S and IS, dimensioned $\geq$ 73, may be physically the same.
SCOPY	N,A,IA,B,IB	Copy N items A(LA), A(LA+IA), ... to B(LB), B(LB+IB), ..., where LA = max (1, 1 - (N - 1) IA) LB = max (1, 1 - (N - 1) IB)

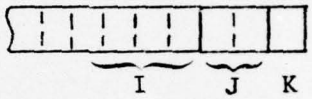
Table B.1 (Continued)

NAME	ARGUMENTS	PURPOSE
SDOT	N,A,IA,B,I,IB	Single precision dot product function of vectors A and B. See SCOPY for item convention.
SELVOP	N,A,X,IX,Y,IY	Replace vector Y by A*X + Y where A is a scalar. See SCOPY for item convention.
SJ2	N, A,IA, B,IB, S,T	Apply Jacobi plane rotation to vectors A and B. See SCOPY for item convention. If a is the rotation angle, then S = sin (a) and T = S/(1 + cos (a)).
SORT	V, N, P	Integer pointer vector P points to items in V in order smallest to largest.
THYME	T, MSG	Determine current time T and print with 30 character message MSG.
TYPCHK	TYPE, LIST, N	Integer function giving index of item in LIST (of length N) which matches TYPE. If no match, print error message and stop.

Every array record consists of two parts: the header and the body. The header contains three parts: the type and the array dimensions. For a vector, the second dimension is 1.

Three bit fields at the right of the type number are used to define the form of the array as indicated in Table B.2.

Table B.2 Array Type Designator



Decimal equivalent 8 I + 2 J + K

K	Interpretation
0	Normal
1	Factored

J	Interpretation
0	Integer
1	Real, single precision
2	Real, double precision
3	Complex

I	Interpretation
0	Full rectangular
3	Diagonal
1,2,4-9	... Not used
10	Profile (skyline) STAGS form at
11-15	...Not used
16	Vector set
17-	... Not used

It is not possible for each matrix-vector utility to handle all possible types (or combination of types) of arrays. Each utility has an internal table of types, corresponding to each array involved, for which it is designed to

operate. Whenever a utility is called, it immediately compares the given types with those in the table in order to determine how to proceed if, indeed, it can proceed. If it cannot, it prints an error message and stops.

After a general matrix-vector utility has sorted out which specific array type or type combination it is to operate on, it usually calls upon a special purpose matrix-vector utility to do the actual work. These support utilities are identified by the final two letters as follows: we consider the assignment of letters to the designator field I in Table B.2 such that A corresponds to I=0, B to I=1, ... , K to I=10, Q to I=16, ... Then, for example, if a utility operates on a type 10 matrix, the last letter in its name will be K. If it operates on types 16 and 0 arrays in that order, its last two letters are QA. Notice the general purpose utilities MPYAA and MTMPAA for full, in core matrices. These utilities do not use a header.

The matrix-vector utilities used in support of SLICE 75 are listed in Table B.3. The array arguments define either source (existing) or destination (to be constructed) arrays. The source arrays are always names of existing records maintained by AID. The destination arrays may be 0 in which case the utility will declare the requred record and provide both the array and its name.

Table B.3 Matrix-Vector Utilities
Supportive of SLICE 75

Note: All arguments are internal
names of records except as noted.

Name	Arguments	Purpose
MFAC	FA, A, CON, B	Set FA to the factorization of A + CON*B, CON a scalar.
MFACK	LB, B1, B2, B3	Integer vector LB provides the block definitions and auxiliary storage locations. B1, B2 and B3 are temporary storage vectors for matrix blocks.
MMPY	P, A, B	Set P = A*B.
MPYAQ	P, A, B	Set vector set P = A*B where B is a vector set and A is full.
MPYKQ	P, LA, A1, B	Integer LA provides the block definitions of matrix A and A1 is a block of scratch storage. Resulting vector set P = A*B.
MPYQA	P, A, B	Vector set P = A*B where A is a vector set and B is full.
MPRINT	A, MSG	Print matrix A with a 10 character message MSG.
MSCL	A, CON, B	Set A = CON*B, CON scalar. A and B may be the same.
MSOL	FA, X, B	Solve A*X = B using factorization FA of A.
MSOLKQ	LA, A1, X, B	Solve vector set equation A*X=B where A is a type 10 matrix with LA the block definition vector and A1 a scratch storage space.
VPRINT	V, MSG	Print vector V with a 10 character message MSG.
VSCL	A, CON, B	Set vector A = CON*B, CON a scalar.
VSUM	A, B, CON, C	Set A = B + CON*C, CON a scalar.

REFERENCES

1. Bathe, K. J., "Solution Methods for Large Generalized Eigenvalue Problems in Structural Engineering," Ph.D. Dissertation, Univ. of California, Berkeley, Calif.

2. Garbow, B. S. and J. J. Donganra, "Path Chart and Documentation for EISPACK Package of Matrix Eigensystem Routines," TM 250, Argonne Nat. Laboratory, Argonne, Ill., July, 1975, updated August, 1975

3. Householder, A. S., The Theory of Matrices in Numerical Analysis, Blaisdell, New York, 1964

4. Jensen, P. S., "The Solution of Large Symmetric Eigenproblems by Sectioning," SIAM J. Numer. Anal., 9, 4 (1972) 534-545

5. _____, "An Inclusion Theorem Related to Inverse Iteration," Linear Algebra and Its Applications, 6 (1973) 209-215

6. _____, "AID, A Storage Manager for Numerical Processes," Lockheed Palo Alto Research Laboratory, Palo Alto, Calif. (1976)

7. Mc Cormick, S. F., "The Sparse Matrix Eigenproblem," Math. Dept., Colorado State Univ., Fort Collins, Colo. (1975)

8. Rutishauser, H., "Computational Aspects of F. I. Bauer's Simultaneous Iteration Method," Numer. Math., 13 (1969) 4-13

9. Stewart, G. W., "Accelerating the Orthogonal Iteration for the Eigenvectors of a Hermitian Matrix," Numer. Math., 13 (1969) 362-376

10. Stewart, G. W., "A Bibliographical Tour of the Large, Sparse Generalized Eigenvalue Problem," in Sparse Matrix Computations (J. R. Bunch and D. J. Rose, eds.), Academix Press, New York (1976)